Add a permission의

# AWS 결제 암호화

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

# AWS 결제 암호화: Add a permission의

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

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 관련하여 고객에게 혼동을 일으킬 수 있는 방식이나 Amazon 브랜드 이미지를 떨어뜨리는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

# Table of Contents

AWS Payment Cryptography란 무엇입니까? ..... 1
개념 ..... 2
업계 용어 ..... 4
공통 키 유형 ..... 4
기타 용어 ..... 7
관련 서비스 ..... 11
자세한 정보 ..... 11
엔드포인트 ..... 12
컨트롤 플레인 엔드포인트 ..... 12
데이터 영역 엔드포인트 ..... 15
시작하기 ..... 18
사전 조건 ..... 18
1단계: 키 생성 ..... 19
2단계: 키를 사용하여 CVV2 값 생성 ..... 20
3단계: 2단계에서 생성된 값 확인 ..... 20
4단계: 네거티브 테스트 수행 ..... 21
5단계: (선택 사항) 정리 ..... 21
키 관리 ..... 23
키 생성 ..... 23
3KEY TDES 기본 파생 키 생성 ..... 24
CVV/CVV2용 2KEY TDES 키 생성 ..... 26
HMAC 키 생성 ..... 27
AES-256 키 생성 ..... 28
PIN 암호화 키(PEK) 생성 ..... 29
비대칭(RSA) 키 생성 ..... 30
PIN 확인 값(PVV) 키 생성 ..... 31
비대칭 ECC 키 생성 ..... 32
키 나열 ..... 33
키 활성화 및 비활성화 ..... 34
키 사용 시작 ..... 34
키 사용 중지 ..... 36
키 복제 ..... 38
다중 리전 키 복제의 이점 ..... 38
다중 리전 키 복제 작동 방식 ..... 38

제한 사항 및 고려 사항 ..... 38
다중 리전 키 복제 활성화 ..... 39
다중 리전 키 복제 비활성화 ..... 41
보안 고려 사항 ..... 42
모범 사례 ..... 42
가격 책정 ..... 42
키 삭제 ..... 42
대기 기간에 대해 ..... 43
키 가져오기 및 내보내기 ..... 47
키 가져오기 ..... 49
키 내보내기 ..... 74
고급 주제 ..... 96
별칭 사용 ..... 107
별칭 정보 ..... 108
애플리케이션에서 별칭 사용 ..... 111
관련 API ..... 111
키 가져오기 ..... 111
키 페어와 연결된 퍼블릭 키/인증서 가져오기 ..... 113
키 태그 지정 ..... 114
AWS Payment Cryptography의 태그 정보 ..... 114
콘솔에서 키 태그 보기 ..... 116
API 작업으로 키 태그 관리 ..... 116
태그에 대한 액세스 제어 ..... 118
태그를 사용하여 키에 대한 액세스 제어 ..... 122
주요 특성 이해 ..... 126
대칭 키 ..... 126
비대칭 키 ..... 128
데이터 작업 ..... 130
데이터 암호화, 복호화 및 재암호화 ..... 130
Encrypt data ..... 131
데이터 해독 ..... 136
카드 데이터 생성 및 확인 ..... 140
카드 데이터 생성 ..... 141
카드 데이터 확인 ..... 142
PIN 데이터 생성, 변환 및 확인 ..... 144
PIN 데이터 변환 ..... 145

PIN 데이터 생성 ..... 147
PIN 데이터 확인 ..... 151
인증 요청(ARQC) 암호화 확인 ..... 155
트랜잭션 데이터 구축 ..... 156
트랜잭션 데이터 패딩 ..... 156
예제 ..... 157
HMAC 생성 및 확인 ..... 158
MAC 생성 ..... 160
MAC 검증 ..... 163
특정 데이터 작업을 위한 키 유형 ..... 165
GenerateCardData ..... 166
VerifyCardData ..... 167
GeneratePinData(VISA/ABA 체계용) ..... 168
GeneratePinData(IBM3624용) ..... 169
VerifyPinData(비자/ABA 체계용) ..... 170
VerifyPinData(IBM3624용) ..... 170
데이터 해독 ..... 171
데이터 암호화 ..... 172
PIN 데이터 변환 ..... 173
MAC 생성/확인 ..... 174
GenerateMacEmvPinChange ..... 176
VerifyAuthRequestCryptogram ..... 177
가져오기/내보내기 키 ..... 177
미사용 키 유형 ..... 178
일반 사용 사례 ..... 179
발급자 및 발급자 프로세서 ..... 179
일반 함수 ..... 179
네트워크별 함수 ..... 199
인수 및 결제 진행자 ..... 223
동적 키 사용 ..... 224
리전별 기능 ..... 227
AS2805 ..... 227
초기 키(KEK) 교환 ..... 229
KEK 검증 ..... 230
작업 키 생성 및 전송 ..... 233
작업 키 내보내기 ..... 235

핀 번역 ..... 235
Mac 생성 및 검증 ..... 237
보안 ..... 238
데이터 보호 ..... 238
키 구성 요소 보호 ..... 240
데이터 암호화 ..... 240
저장 시 암호화 ..... 240
전송 중 암호화 ..... 240
인터네트워크 트래픽 개인 정보 ..... 241
복원력 ..... 241
리전별 격리 ..... 242
멀티 테넌트 디자인 ..... 242
인프라 보안 ..... 243
물리적 호스트의 격리 ..... 243
Amazon VPC 및 AWS PrivateLink 사용 ..... 243
AWS Payment Cryptography VPC 엔드포인트에 대한 고려 사항 ..... 244
AWS Payment Cryptography용 VPC 엔드포인트 생성 ..... 245
VPC 엔드포인트에 연결 ..... 245
VPC 엔드포인트에 대한 액세스 제어 ..... 246
정책 설명에 VPC 엔드포인트 사용 ..... 250
VPC 엔드포인트 로깅 ..... 253
하이브리드 포스트 양자 TLS ..... 256
포스트 양자 TLS 소개 ..... 257
PQC 정보 ..... 257
사용 방법 ..... 257
보안 모범 사례 ..... 261
규정 준수 확인 ..... 263
서비스 규정 준수 ..... 263
PIN 규정 준수 ..... 264
공통 주제 ..... 264
평가 범위 ..... 266
트랜잭션 처리 작업 ..... 268
P2PE 규정 준수 ..... 272
ID 및 액세스 관리 ..... 274
대상 ..... 274
ID를 통한 인증 ..... 274

AWS 계정 루트 사용자 ..... 275
IAM 사용자 및 그룹 ..... 275
IAM 역할 ..... 275
정책을 사용하여 액세스 관리 ..... 275
ID 기반 정책 ..... 276
리소스 기반 정책 ..... 276
액세스 제어 목록(ACL) ..... 276
기타 정책 타입 ..... 277
여러 정책 유형 ..... 277
AWS Payment Cryptography가 IAM과 작동하는 방식 ..... 277
AWS Payment Cryptography 자격 증명 기반 정책 ..... 277
AWS Payment Cryptography 태그 기반 권한 부여 ..... 279
ID 기반 정책 예시 ..... 280
정책 모범 사례 ..... 280
콘솔 사용 ..... 281
사용자가 자신이 권한을 볼 수 있도록 허용 ..... 281
AWS Payment Cryptography의 모든 측면에 액세스할 수 있는 기능 ..... 282
지정된 키를 사용하여 API 호출 가능 ..... 283
리소스를 구체적으로 거부할 수 있음 ..... 284
리소스 기반 정책 ..... 285
고려 사항 ..... 286
리소스 기반 정책 관리 ..... 287
리소스 기반 정책 예제 ..... 288
다자간 승인 ..... 289
개요 ..... 290
보호된 작업 ..... 291
사전 조건 ..... 291
MPA 활성화 및 비활성화 ..... 291
시작하기 ..... 292
예: MPA가 활성화된 루트 인증서 가져오기 ..... 292
AWS CloudTrail MPA 이벤트에 대한 로깅 ..... 293
요청 상태 확인 및 실패 처리 ..... 296
문제 해결 ..... 298
모니터링 ..... 299
CloudTrail 로그 ..... 299
AWS CloudTrail의 Payment Cryptography 정보 ..... 300

CloudTrail의 컨트롤 플레인 이벤트 ..... 301
CloudTrail의 데이터 이벤트 ..... 301
AWS Payment Cryptography Control Plane 로그 파일 항목 이해 ..... 302
AWS Payment Cryptography 데이터 영역 로그 파일 항목 이해 ..... 305
암호화 세부 정보 ..... 308
설계 목표 ..... 309
기본 ..... 310
암호 프리미티브 ..... 310
엔트로피 및 난수 생성 ..... 310
대칭 키 작업 ..... 310
비대칭 키 작업 ..... 311
키 스토리지 ..... 311
대칭 키를 사용한 키 가져오기 ..... 312
비대칭 키를 사용한 키 가져오기 ..... 312
키 내보내기 ..... 312
트랜잭션별 파생된 고유 키(DUKPT) 프로토콜 ..... 312
키 계층 구조 ..... 312
내부 작업 ..... 315
HSM 보호 ..... 316
일반 키 관리 ..... 318
고객 키 관리 ..... 321
통신 보안 ..... 323
로깅 및 모니터링 ..... 324
고객 작업 ..... 324
키 생성 ..... 325
키 가져오기 ..... 325
키 내보내기 ..... 326
키 삭제 ..... 326
키 교체 중 ..... 326
할당량: ..... 327
문서 이력 ..... 329
..... cccxxxi

# AWS Payment Cryptography란 무엇입니까?

AWS Payment Cryptography는 전용 결제 HSM 인스턴스를 조달할 필요 없이 결제 카드 산업(PCI) 표준에 따라 결제 처리에 사용되는 암호화 함수 및 키 관리에 대한 액세스를 제공하는 관리형 AWS 서비스입니다. AWS Payment Cryptography는 인수자와 같은 결제 기능을 수행하는 고객에게 결제 진행자 네트워크, 스위치, 프로세서, 및 은행은 결제 암호화 작업을 클라우드의 애플리케이션에 더 가깝게 이동하고 전용 결제 HSMs.

이 서비스는 PCI PIN, PCI P2PE 및 PCI DSS를 포함한 해당 업계 규칙을 준수하도록 설계되었으며, 이 서비스에서 활용하는 하드웨어는 PCI PTS HSM V3 및 FIPS 140-2 레벨 3 인증을 받았습니다. 짧은 지연 시간과 높은 수준의 가동 시간 및 복원력을 지원하도록 설계되었습니다. AWS Payment Cryptography는 완전히 탄력적이며 하드웨어 프로비저닝, 키 구성 요소 보안 관리, 보안 시설에 긴급 백업 유지 관리 등 온프레미스 HSMs의 많은 운영 요구 사항을 제거합니다. AWS 또한 Payment Cryptography는 파트너와 전자적으로 키를 공유할 수 있는 옵션을 제공하므로 종이로 된 일반 텍스트 구성 요소를 공유할 필요가 없습니다.

AWS Payment Cryptography 컨트롤 플레인 API를 사용하여 키를 생성하고 관리할 수 있습니다.

AWS Payment Cryptography 데이터 영역 API를 사용하여 결제 관련 트랜잭션 처리 및 관련 암호화 작업에 암호화 키를 사용할 수 있습니다.

AWS Payment Cryptography는 키를 관리하는 데 사용할 수 있는 중요한 기능을 제공합니다.

- TDES, AES 및 RSA 키를 포함한 대칭 및 비대칭 AWS Payment Cryptography 키를 생성 및 관리하고 CVV 생성 또는 DUKPT 키 추출과 같은 의도한 목적을 지정합니다.
- AWS Payment Cryptography 키를 하드웨어 보안 모듈(HSMs)로 보호하면서 사용 사례 간에 키 분리를 적용하면서 자동으로 안전하게 저장합니다.
- AWS Payment Cryptography 키에 액세스하거나 액세스를 제어하는 데 사용할 수 있는 “표시된 이름”인 별칭을 생성, 삭제, 나열 및 업데이트합니다.
- 식별, 그룹화, 자동화, 액세스 제어 및 비용 추적을 위해 AWS Payment Cryptography 키에 태그를 지정합니다.
- TR-31(상호 운용 가능한 보안 키 교환 키 블록 사양)에 따라 키 암호화 키(KEK)를 사용하여 AWS Payment Cryptography와 HSM(또는 타사) 간에 대칭 키를 가져오고 내보냅니다.
- TR-34(비대칭 기법을 사용한 대칭 키 배포 방법)와 같은 전자적 수단을 사용하여 다음에 비대칭 키 페어를 사용하여 AWS Payment Cryptography와 기타 시스템 간에 대칭 키 암호화 키(KEK)를 가져오고 내보냅니다.

Payment AWS Cryptography 키를 다음과 같은 암호화 작업에 사용할 수 있습니다.

- 대칭 또는 비대칭 AWS Payment Cryptography 키를 사용하여 데이터를 암호화, 복호화 및 다시 암호화합니다.
- PCI PIN 규칙에 따라 일반 텍스트를 노출하지 않고 암호화 키 간에 민감한 데이터(예: 카드 소유자 핀)를 안전하게 변환합니다.
- CVV, CVV2 또는 ARQC와 같은 카드 소유자 데이터를 생성하거나 검증합니다.
- 카드 소유자 핀을 생성하고 검증합니다.
- MAC 서명을 생성하거나 검증합니다.

# 개념

AWS Payment Cryptography에 사용되는 기본 용어 및 개념과 이를 사용하여 데이터를 보호하는 방법을 알아봅니다.

별칭

AWS Payment Cryptography 키와 연결된 사용자 친화적 이름입니다. 별칭은 많은 AWS Payment Cryptography API 작업에서 키 ARN과 상호 교환적으로 사용할 수 있습니다. 별칭을 사용하면 애플리케이션 코드에 영향을 주지 않고 키를 교체하거나 변경할 수 있습니다. 별칭 이름은 최대 256자의 문자열입니다. 계정 및 리전 내에서 연결된 AWS Payment Cryptography 키를 고유하게 식별합니다. AWS Payment Cryptography에서 별칭 이름은 항상 로 시작합니다`alias/`.

별칭 이름의 형식은 다음과 같습니다.

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

예시:

```
alias/sampleAlias2
```

키 ARN

키 ARN은 AWS Payment Cryptography에 있는 키 항목의 Amazon 리소스 이름(ARN)입니다. Payment AWS Cryptography 키의 고유하고 정규화된 식별자입니다. 키 ARN에는 AWS 계정, 리전 및 무작위로 생성된 ID가 포함됩니다. ARN은 키 구성 요소와 관련되거나 파생되지 않습니다. 생성 또는 가져오기 작업 중에 자동으로 할당되기 때문에 이러한 값은 멱등성이 없습니다. 동일한 키를 여러 번 가져오면 고유한 수명 주기를 가진 키 ARN이 여러 개 생성됩니다.

키 ARN의 형식은 다음과 같습니다.

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

다음은 키 ARN 샘플입니다.

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

키 식별자

키 식별자는 키에 대한 참조이며이 중 하나(또는 그 이상)는 AWS Payment Cryptography 작업에 대한 일반적인 입력입니다. 유효한 키 식별자는 키 Arn a Key 별칭일 수 있습니다.

AWS Payment Cryptography 키

AWS Payment Cryptography 키(키)는 모든 암호화 함수에 사용됩니다. 키는 create key 명령을 사용하여 직접 생성하거나 키 가져오기를 호출하여 시스템에 추가할 수 있습니다. 키의 오리진은 KeyOrigin 속성을 검토하여 확인할 수 있습니다. AWS Payment Cryptography는 DUKPT에서 사용하는 키와 같이 암호화 작업 중에 사용되는 파생 키 또는 중간 키도 지원합니다.

이러한 키는 생성 시 정의된 변경 불가 속성과 변경 가능 속성을 모두 가지고 있습니다. 알고리즘, 길이, 사용 등의 속성은 생성 시 정의되며 변경할 수 없습니다. 발효일 또는 만료일과 같은 기타 항목은 수정할 수 있습니다. AWS Payment Cryptography 키 속성의 전체 목록은 Payment Cryptography API 참조를 참조하세요. AWS

AWS Payment Cryptography 키에는 주로 ANSI X9 TR 31에 의해 정의된 키 유형이 있으며, 이는 PCI PIN v3.1 요구 사항 19에 지정된 대로 의도한 용도로 사용을 제한합니다.

속성은 PCI PIN v3.1 요구 사항 18-3에 지정된 대로 저장하거나, 다른 계정과 공유하거나, 내보낼 때 키 블록을 사용하여 키에 바인딩됩니다.

키는 키 Amazon 리소스 이름()이라고 하는 고유한 값을 사용하여 AWS Payment Cryptography 플랫폼에서 식별됩니다ARN.

> **Note**
>
> 키ARN는 키를 처음 생성하거나 AWS Payment Cryptography 서비스로 가져올 때 생성됩니다. 따라서 키 가져오기 기능을 사용하여 동일한 키 자료를 여러 번 추가하면 동일한 키 구성 요소가 여러 개의 키 ARNS 아래에 위치하지만 각각의 수명 주기는 다릅니다.

# 업계 용어

주제


- 공통 키 유형
- 기타 용어


## 공통 키 유형

AWS Payment Cryptography 키

AWS Payment Cryptography 키는 단일에 있습니다 AWS 리전. Payment AWS Cryptography Service에 저장된 주요 메타데이터와 자료로 구성됩니다. 키는 외부 소스에서 TR-31 키 블록으로 가져오거나 AWS Payment Cryptography Service에서 생성할 수 있습니다.

AWK

취득자 작업 키(AWK)는 일반적으로 취득자/취득자 프로세서와 네트워크(예: Visa 또는 Mastercard) 간에 데이터를 교환하는 데 사용되는 키입니다. 지금까지 AWK는 암호화에 3DES를 활용하고 이를 TR31_P0_PIN_ENCRYPTION_KEY로 표현했습니다.

BDK

기본 파생 키(BDK)는 후속 키를 도출하는 데 사용되는 작업 키이며 일반적으로 PCI PIN 및 PCI P2PE DUKPT 프로세스의 일부로 사용됩니다. TR31_B0_BASE_DRIVATIVATION_KEY로 표시됩니다.

CMK

카드 마스터 키(CMK)는 일반적으로 발급자 마스터 키, PAN 및 PSN에서 파생된 하나 이상의 카드별 키(들)이며 일반적으로 3DES 키입니다. 이러한 키는 개인화 과정에서 EMV 칩에 저장됩니다. CMK의 예로는 AC, SMI 및 SMC 키가 있습니다.

CMK-AC

애플리케이션 암호(AC) 키는 EMV 트랜잭션의 일부로 트랜잭션 암호를 생성하는 데 사용되며 일종의 카드 마스터 키입니다.

CMK-SMI

보안 메시징 무결성(SMI) 키는 EMV의 일부로 사용되어 핀 업데이트 스크립트와 같은 MAC을 사용하여 카드로 전송되는 페이로드의 무결성을 확인합니다. 일종의 카드 마스터 키입니다.

CMK-SMC

보안 메시징 기밀성(SMC) 키는 EMV의 일부로 핀 업데이트와 같이 카드로 전송되는 데이터를 암호화하는 데 사용됩니다. 일종의 카드 마스터 키입니다.

CVK

카드 검증 키(CVK)는 정의된 알고리즘을 사용하여 CVV, CVV2 및 유사한 값을 생성하고 입력을 검증하는 데 사용되는 키입니다. TR31_C0_CARD_VERIFICATION_KEY로 표시됩니다.

IMK

발급자 마스터 키(IMK) 는 EMV 칩 카드 개인화의 일부로 사용되는 마스터 키입니다. 일반적으로 3개의 IMK가 있으며, 각각 AC(암호문), SMI(무결성/서명을 위한 스크립트 마스터 키), SMC(기밀성/암호화를 위한 스크립트 마스터 키) 키에 하나씩 사용됩니다.

IK

초기 키(IK)는 DUKPT 프로세스에 사용되는 첫 번째 키이며 기본 파생 키(BDK)에서 파생됩니다. 이 키에 대한 트랜잭션은 처리되지 않지만 트랜잭션에 사용될 미래 키를 도출하는 데 사용됩니다. IK를 생성하기 위한 파생 방법은 X9.24-1:2017에 정의되었습니다. TDES BDK를 사용하는 경우 X9.24-1:2009가 적용 가능한 표준이며 IK는 초기 핀 암호화 키(IPEK)로 대체됩니다.

IPEK

초기 PIN 암호화 키(IPEK)는 DUKPT 프로세스에 사용되는 초기 키이며 기본 파생 키(BDK)에서 파생됩니다. 이 키에 대한 트랜잭션은 처리되지 않지만 트랜잭션에 사용될 미래 키를 도출하는 데 사용됩니다. 이 키는 데이터 암호화 및 mac 키를 도출하는 데에도 사용할 수 있으므로 IPEK는 잘못된 것입니다. IPEK를 생성하기 위한 파생 방법은 X9.24-1:2009에 정의되어 있습니다. AES BDK를 사용하는 경우 X9.24-1:2017이 적용 가능한 표준이며 IPEK는 초기 키(IK)로 대체됩니다.

IWK

발급자 작업 키(IWK)는 일반적으로 발급자/발급자 프로세서와 네트워크(예: Visa 또는 Mastercard) 간에 데이터를 교환하는 데 사용되는 키입니다. 지금까지 IWK는 암호화에 3DES를 활용하고 TR31_P0_PIN_ENCRYPTION_KEY로 표시했습니다.

KBPK

키 블록 암호화 키(KBPK)는 키 블록을 보호하여 다른 키를 래핑/암호화하는 데 사용되는 대칭 키 유형입니다. KBPK는 KEK와 유사하지만 KEK는 키 구성 요소를 직접 보호하는 반면 TR-31 및 유사한 체계에서는 KBPK가 작동 키만 간접적으로 보호합니다. TR-31를 사용하는 경우 TR31_K1_KEY_BLOCK_PROTECTION_KEY가 올바른 키 유형이지만 TR31_K0_KEY_ENCRYPTION_KEY는 기록 목적으로 상호 교환적으로 지원됩니다.

KEK

키 암호화 키(KEK)는 전송 또는 저장을 위해 다른 키를 암호화하는 데 사용되는 키 입니다. 다른 키를 보호하기 위한 키는 일반적으로 TR-31 표준에 따라 KeyUsage가 TR31_K0_KEY_ENCRYPTION_KEY입니다.

PEK

PIN 암호화 키(PEK)는 두 당사자 간의 저장 또는 전송을 위해 PIN을 암호화하는 데 사용되는 일종의 작업 키입니다. IWK와 AWK는 핀 암호화 키를 구체적으로 사용하는 두 가지 예입니다. 이러한 키는 TR31_P0_PIN_ENCRYPTION_KEY로 표시됩니다.

PGK

PGK(핀 생성 키)는 핀 확인 키의 또 다른 이름입니다. 실제로는 핀(기본적으로 암호화 방식으로 난수)을 생성하는 데 사용되지 않지만 PVV와 같은 확인 값을 생성하는 데 사용됩니다.

PRK

기본 리전 키는 복제가 활성화된 지정된 Payment Cryptography 키의 신뢰할 수 있는 복제 소스입니다. PRK는 다중 리전 키 복제 구성의 소스 Payment Cryptography 키 역할에 대한 참조입니다. Payment Cryptography 키에서 복제가 활성화되면 해당 특정 키 복제 구성에 대한 PRK라고 합니다.

PVK

PIN 검증 키(PVK)는 PVV와 같은 PIN 확인 값을 생성하는 데 사용되는 작업 키 유형입니다. 가장 일반적인 두 가지 종류는 IBM3624 오프셋 값을 생성하는 데 사용되는 TR31_V1_IBM3624_PIN_VERIFICATION_KEY와 Visa/ABA 검증 값에 사용되는 TR31_V2_VISA_PIN_VERIFICATION_KEY입니다. 이를 핀 생성 키라고도 합니다.

RRK

복제본 리전 키는 PRK에서 구성된 복제본으로 안전하게 복사되는 복제된 키 구성 요소 및 메타데이터입니다 AWS 리전. RRK는 Payment Cryptography 키의 읽기 전용 복제본입니다. RRK는 다중 리전 키 복제 구성에서 특정 키가 수행하는 역할을 참조합니다. 복제 설정을 포함한 모든 주요 메타데이터 변경 사항을 PRK에 적용해야 합니다.

# 기타 용어

ARQC

승인 요청 암호문(ARQC)은 EMV 표준 칩 카드(또는 이와 동등한 비접촉식 구현)를 통해 트랜잭션 시점에 생성되는 암호문입니다. 일반적으로 ARQC는 칩 카드로 생성되며 발급자 또는 해당 에이전트에게 전달되어 트랜잭션 시 확인됩니다.

CVV

카드 확인 값은 전통적으로 마그네틱 스트라이프에 포함되었으며 트랜잭션의 신뢰성을 검증하는 데 사용된 정적 보안 암호 값입니다. 알고리즘은 iCVV, CAVV, CVV2와 같은 다른 용도로도 사용됩니다. 다른 사용 사례에서는 이러한 방식으로 내장되지 않을 수 있습니다.

CVV2

카드 확인 값 2는 전통적으로 결제 카드의 전면(또는 후면)에 인쇄되었으며 카드가 없는 결제(예: 전화 또는 온라인)의 신뢰성을 확인하는 데 사용되는 정적 보안 암호 값입니다. CVV와 동일한 알고리즘을 사용하지만 서비스 코드는 000으로 설정됩니다.

iCVV

iCVV는 CVV2-like 값이지만 EMV(Chip) 카드의 track2에 상응하는 데이터에 포함되어 있습니다. 이 값은 서비스 코드 999를 사용하여 계산되며 CVV1/CVV2와 다르므로 도난 정보가 다른 유형의 새 결제 자격 증명을 생성하는 데 사용되지 않습니다. 예를 들어 칩 트랜잭션 데이터를 가져온 경우이 데이터를 사용하여 마그네틱 스트라이프(CVV1) 또는 온라인 구매(CVV2)를 생성할 수 없습니다.

??? 키를 사용합니다.

DUKPT

트랜잭션별 파생된 고유 키(DUKPT)는 일반적으로 물리적 POS/POI에서 일회용 암호화 키 사용을 정의하는 데 사용되는 키 관리 표준입니다. 지금까지 DUKPT는 3DES를 암호화에 활용했습니다. DUKPT의 업계 표준은 ANSI X9.24-3-2017에 정의되어 있습니다.

ECC

ECC(Elliptic Curve Cryptography)는 타원 곡선의 수학을 사용하여 암호화 키를 생성하는 퍼블릭 키 암호화 시스템입니다. ECC는 RSA와 같은 기존 방법과 동일한 보안 수준을 제공하지만 키 길이가 훨씬 짧아 보다 효율적인 방식으로 동등한 보안을 제공합니다. 이는 RSA가 실용적인 솔루션이 아닌 사용 사례(RSA 키 길이 > 4096비트)와 특히 관련이 있습니다. AWS Payment Cryptography는 ECDH 작업에 사용하기 위해 NIST에서 정의한 곡선을 지원합니다.

ECDH

ECDH(Elliptic Curve Diffie-Hellman)는 두 당사자가 공유 보안 암호(예: KEK 또는 PEK)를 설정할 수 있도록 허용하는 주요 계약 프로토콜입니다. ECDH에서 당사자 A와 B는 각각 고유한 퍼블릭-프라이빗 키 페어를 가지고 있으며, 서로 퍼블릭 키( AWS Payment Cryptography용 인증서 형식)와 키 파생 메타데이터(파생 방법, 해시 유형 및 공유 정보)를 교환합니다. 양 당사자는 프라이빗 키에 상대방의 퍼블릭 키를 곱하고 타원 곡선 속성으로 인해 결과 키를 추출(생성)할 수 있습니다.

EMV

EMV(원래 Europay, Mastercard, Visa)는 결제 이해관계자와 협력하여 상호 운용 가능한 결제 표준 및 기술을 생성하는 기술 기관입니다. 한 가지 표준 예는 칩/비접촉 카드와 사용된 암호화를 포함하여 상호 작용하는 결제 터미널에 대한 것입니다. EMV 키 유도는와 같은 초기 키 세트를 기반으로 각 결제 카드에 대해 고유한 키를 생성하는 방법(들)을 의미합니다. IMK

HSM

하드웨어 보안 모듈(HSM)은 암호화 작업(예: 암호화, 해독, 디지털 서명)과 이러한 작업에 사용되는 기본 키를 보호하는 물리적 장치입니다.

KCAAS

Key Custodian As A Service(KCAAS)는 키 관리와 관련된 다양한 서비스를 제공합니다. 결제 키의 경우 일반적으로 종이 기반 키 구성 요소를 AWS Payment Cryptography에서 지원하는 전자 양식으로 변환하거나 전자적으로 보호된 키를 특정 공급업체에 필요할 수 있는 종이 기반 구성 요소로 변환할 수 있습니다. 또한 손실이 진행 중인 작업에 해를 끼칠 수 있는 키에 대한 키 에스크로 서비스를 제공할 수 있습니다. KCAAS 공급업체는 고객이 PCI DSS, PCI PIN 및 PCI P2PE 표준을 준수하는 방식으로 AWS Payment Cryptography와 같은 보안 서비스 외부에서 키 구성 요소를 관리하는 운영 부담을 덜 수 있도록 지원할 수 있습니다. AWS Payment Cryptography는 종이 기반 키 구성 요소를 전자 형식으로 변환하기 위한 내장 KCAAS 기능으로 물리적 키 교환를 제공합니다.

KCV

키 확인 값(KCV)은 실제 키 자료에 액세스하지 않고도 키를 서로 비교하는 데 주로 사용되는 다양한 체크섬 방법을 말합니다. KCV는 무결성 검증(특히 키 교환 시)에도 사용되었지만, 이제 이 역할은 TR-31와 같은 키 블록 형식의 일부로 포함되어 있습니다. TDES 키의 경우 KCV는 검사할 키와 함께 각각 값이 0인 8바이트를 암호화하고 암호화된 결과 중 가장 높은 순위의 3바이트를 유지하는 방식으로 계산됩니다. AES 키의 경우 KCV는 입력 데이터가 0의 16바이트이고 암호화된 결과 중 가장 높은 순위의 3바이트를 유지하는 CMAC 알고리즘을 사용하여 계산됩니다.

KDH

키 분포 호스트(KDH)는 TR-34와 같은 키 교환 프로세스를 통해 키를 전송하는 장치 또는 시스템입니다. AWS Payment Cryptography에서 키를 보내는 경우 이를 KDH로 간주합니다.

KIF

키 삽입 기능(KIF)은 암호화 키를 로드하는 것을 포함하여 결제 단말기를 초기화하는 데 사용되는 보안 시설입니다.

KRD

키 수신 장치(KRD)는 TR-34와 같은 키 교환 프로세스에서 키를 수신하는 장치입니다. AWS Payment Cryptography로 키를 전송할 때 키를 KRD로 간주합니다.

KSN

키 일련 번호(KSN)는 트랜잭션별 고유한 암호화 키를 생성하기 위해 DUKPT 암호화/해독에 입력 값으로 사용되는 값입니다. KSN은 일반적으로 BDK 식별자, 준고유 단말기 ID, 특정 결제 단말기에서 전환이 처리될 때마다 증가하는 트랜잭션 카운터로 구성됩니다. X9.24에 따라 TDES의 경우 10바이트 KSN은 일반적으로 키 세트 ID의 경우 24비트, 터미널 ID의 경우 19비트, 트랜잭션 카운터의 경우 21비트로 구성되지만 키 세트 ID와 터미널 ID 사이의 경계는 AWS Payment Cryptography의 기능에 영향을 미치지 않습니다. AES의 경우 12바이트 KSN은 일반적으로 BDK ID의 경우 32비트, 파생 식별자(ID)의 경우 32비트, 트랜잭션 카운터의 경우 32비트로 구성됩니다.

MPoC

MPoC(상용 하드웨어의 모바일 판매 시점)는 판매자가 스마트폰 또는 기타 상용 off-the-shelf품(COTS) 모바일 디바이스를 사용하여 카드 소지자 PINs 또는 비대면 결제를 수락할 수 있는 솔루션의 보안 요구 사항을 해결하는 PCI 표준입니다.

PAN

기본 계좌 번호(PAN)는 신용카드나 직불카드와 같은 계좌의 고유 식별자입니다. 일반적으로 길이는 13-19자리입니다. 처음 6~8자리는 네트워크와 발급 은행을 식별합니다.

PIN 블록

처리 또는 전송 중에 PIN을 포함한 기타 데이터 요소를 포함하는 데이터 블록입니다. PIN 블록 형식은 PIN 블록의 내용과 PIN 블록을 처리하여 PIN을 검색하는 방법을 표준화합니다. 대부분의 PIN 블록은 PIN, PIN 길이로 구성되며 PAN의 일부 또는 전부를 포함하는 경우가 많습니다. AWS Payment Cryptography는 ISO 9564-1 형식 0, 1, 3 및 4를 지원합니다. AES 키에는 형식 4가 필요합니다. PIN을 확인하거나 변환할 때는 수신 또는 발신 데이터의 PIN 블록을 지정해야 합니다.

POI

POS(판매 시점)와 함께 익명으로 자주 사용되는 POI(상호 작용 시점)는 카드 소지자가 결제 자격 증명을 제시하기 위해 상호 작용하는 하드웨어 디바이스입니다. POI의 예로는 가맹점에 있는 실제 단말기가 있습니다. 인증된 PCI PTS POI 단말기 목록은 PCI 웹사이트를 참조하세요.

PSN

PAN 시퀀스 번호(PSN)는 동일한 PAN으로 발급된 여러 카드를 구분하는 데 사용되는 숫자 값입니다.

퍼블릭 키

비대칭 암호(RSA, ECC)를 사용하는 경우 퍼블릭 키는 퍼블릭-프라이빗 키 페어의 퍼블릭 구성 요소입니다. 퍼블릭-프라이빗 키 페어의 소유자에 대한 데이터를 암호화해야 하는 엔터티에 퍼블릭 키를 공유하고 배포할 수 있습니다. 디지털 서명 작업의 경우 서명을 확인하는 데 퍼블릭 키가 사용됩니다.

프라이빗 키

비대칭 암호(RSA,ECC)를 사용하는 경우 프라이빗 키는 퍼블릭-프라이빗 키 페어의 프라이빗 구성 요소입니다. 프라이빗 키는 데이터를 복호화하거나 디지털 서명을 생성하는 데 사용됩니다. 대칭 AWS Payment Cryptography 키와 마찬가지로 프라이빗 키는 HSMs에서 안전하게 생성됩니다. HSM의 휘발성 메모리에만 해독되며 암호화 요청을 처리하는 데 필요한 시간 동안만 해독됩니다.

PVV

PIN 확인 값(PVV)은 실제 핀을 저장하지 않고 핀을 확인하는 데 사용할 수 있는 암호화 출력 유형입니다. 일반 용어이지만 AWS Payment Cryptography의 맥락에서 PVV는 Visa 또는 ABA PVV 메서드를 나타냅니다. 이 PVV는 카드 번호, 팬 시퀀스 번호, 팬 자체 및 PIN 확인 키 입력이 있는 4자리 숫자입니다. 검증 단계에서 AWS Payment Cryptography는 트랜잭션 데이터를 사용하여 PVV를 내부적으로 다시 생성하고 AWS Payment Cryptography 고객이 저장한 값을 다시 비교합니다. 이러한 방식으로 개념적으로 암호화 해시 또는 MAC와 유사합니다.

RSA 래핑/언래핑

RSA 래핑은 비대칭 키를 사용하여 다른 시스템으로 전송하기 위한 대칭 키(예: TDES 키)를 래핑합니다. 일치하는 프라이빗 키가 있는 시스템만 페이로드를 해독하고 대칭 키를 로드할 수 있습니다. 반대로 RSA 언래핑은 RSA를 사용하여 암호화된 키를 안전하게 해독한 다음 키를 AWS Payment Cryptography에 로드합니다. RSA 래핑은 키를 교환하는 하위 수준 방법이며 키 블록 형식으로 키를 전송하지 않으며 전송자의 페이로드 서명을 활용하지 않습니다. 제공 여부 및 키 속성이 변경되지 않았는지 확인하기 위해 대체 제어를 고려해야 합니다.

또한 TR-34는 RSA를 내부적으로 활용하지만 별도의 형식이며 상호 운용할 수 없습니다.

TR-31

TR-31(정식 명칭은 ANSI X9 TR 31)는 키 데이터 자체와 동일한 데이터 구조에서 키 속성을 정의할 수 있도록 미국국립표준협회(ANSI)에서 정의한 키 블록 형식입니다. TR-31 키 블록 형식은 키에 연결된 키 속성 세트를 정의하여 함께 유지됩니다. AWS 결제 암호화는 가능한 경우 TR-31 표준화된 용어를 사용하여 적절한 키 분리 및 키 목적을 보장합니다. TR-31은 ANSI X9.143-2022로 대체되었습니다.

TR-34

TR-34는 비대칭 기법(예: RSA)을 사용하여 대칭 키(예: 3DES 및 AES)를 안전하게 배포하는 프로토콜을 설명하는 ANSI X9.24-2의 구현입니다. AWS Payment Cryptography는 TR-34 메서드를 사용하여 키의 안전한 가져오기 및 내보내기를 허용합니다.

X9.143

X9.143은 동일한 데이터 구조에서 키 및 키 속성을 보호하기 위해 미국 국립 표준 연구소(ANSI)에서 정의한 키 블록 형식입니다. 키 블록 형식은 키에 연결된 키 속성 세트를 정의하여 함께 유지됩니다. AWS 결제 암호화는 적절한 키 분리 및 키 목적을 보장하기 위해 가능하면 X9.143 표준화된 용어를 사용합니다. X9.143은 이전 TR-31 제안을 대체하지만 대부분의 경우 이전 버전과 이전 버전과 호환되며 용어는 종종 상호 교환적으로 사용됩니다.

## 관련 서비스

AWS Key Management Service

AWS Key Management Service(AWS KMS)는 데이터를 보호하는 데 사용되는 암호화 키를 쉽게 생성하고 제어할 수 있는 관리형 서비스입니다. AWS KMS는 하드웨어 보안 모듈(HSMs)을 사용하여 AWS KMS 키를 보호하고 검증합니다.

AWS CloudHSM

AWS CloudHSM 는 AWS 클라우드에서 전용 범용 HSM 인스턴스를 고객에게 제공합니다.는 키 생성, 데이터 서명 또는 데이터 암호화 및 복호화와 같은 다양한 암호화 기능을 제공할 AWS CloudHSM 수 있습니다.

## 자세한 정보

- AWS Payment Cryptography에 사용되는 용어 및 개념에 대한 자세한 내용은 AWS Payment Cryptography 개념을 참조하세요.

- AWS Payment Cryptography Control Plane API에 대한 자세한 내용은 AWS Payment Cryptography Control Plane API 참조를 참조하세요.
- AWS Payment Cryptography Data Plane API에 대한 자세한 내용은 AWS Payment Cryptography Data Plane API 참조를 참조하세요.
- AWS Payment Cryptography가 암호화를 사용하고 AWS Payment Cryptography 키를 보호하는 방법에 대한 자세한 기술 정보는 암호화 세부 정보를 참조하세요.

# 에 대한 엔드포인트 AWS Payment Cryptography

프로그래밍 방식으로에 연결하려면 엔드포인트 AWS Payment Cryptography, 즉 서비스에 대한 진입점의 URL을 사용합니다. AWS SDKs 및 명령줄 도구는 요청의 리전 컨텍스트에 AWS 리전 따라의 서비스에 대한 기본 엔드포인트를 자동으로 사용하므로 일반적으로 이러한 값을 명시적으로 설정할 필요가 없습니다. 필요한 경우 API 요청에 대해 다른 엔드포인트를 지정할 수 있습니다.

## 컨트롤 플레인 엔드포인트

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 | |
|---|---|---|---|---|
| 미국 동부(오하이오) | us-east-2 | controlplane.payment-cryptography.us-east-2.amazonaws.com<br><br>controlplane.payment-cryptography.us-east-2.api.aws | HTTPS<br><br>HTTPS | |
| 미국 동부(버지니아 북부) | us-east-1 | controlplane.payment-cryptography.us-east-1.amazonaws.com<br><br>controlplane.payment-cryptography.us-east-1.api.aws | HTTPS<br><br>HTTPS | |
| 미국 서부(오리곤) | us-west-2 | controlplane.payment-cryptography.us-west-2.amazonaws.com<br><br>controlplane.payment-cryptography.us-west-2.api.aws | HTTPS<br><br>HTTPS | |

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 |
| --- | --- | --- | --- |
| 아프리카(케이프타운) | af-south-1 | controlplane.payment-cryptography.af-south-1.amazonaws.com<br>controlplane.payment-cryptography.af-south-1.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(하이데라바드) | ap-south-2 | controlplane.payment-cryptography.ap-south-2.amazonaws.com<br>controlplane.payment-cryptography.ap-south-2.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(뭄바이) | ap-south-1 | controlplane.payment-cryptography.ap-south-1.amazonaws.com<br>controlplane.payment-cryptography.ap-south-1.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(오사카) | ap-northeast-3 | controlplane.payment-cryptography.ap-northeast-3.amazonaws.com<br>controlplane.payment-cryptography.ap-northeast-3.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(싱가포르) | ap-southeast-1 | controlplane.payment-cryptography.ap-southeast-1.amazonaws.com<br>controlplane.payment-cryptography.ap-southeast-1.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(시드니) | ap-southeast-2 | controlplane.payment-cryptography.ap-southeast-2.amazonaws.com<br>controlplane.payment-cryptography.ap-southeast-2.api.aws | HTTPS<br>HTTPS |

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 |
| --- | --- | --- | --- |
| 아시아 태평양(도쿄) | ap-northeast-1 | controlplane.payment-cryptography.ap-northeast-1.amazonaws.com<br>controlplane.payment-cryptography.ap-northeast-1.api.aws | HTTPS<br>HTTPS |
| 캐나다(중부) | ca-central-1 | controlplane.payment-cryptography.ca-central-1.amazonaws.com<br>controlplane.payment-cryptography.ca-central-1.api.aws | HTTPS<br>HTTPS |
| 유럽(프랑크푸르트) | eu-central-1 | controlplane.payment-cryptography.eu-central-1.amazonaws.com<br>controlplane.payment-cryptography.eu-central-1.api.aws | HTTPS<br>HTTPS |
| 유럽(아일랜드) | eu-west-1 | controlplane.payment-cryptography.eu-west-1.amazonaws.com<br>controlplane.payment-cryptography.eu-west-1.api.aws | HTTPS<br>HTTPS |
| 유럽(런던) | eu-west-2 | controlplane.payment-cryptography.eu-west-2.amazonaws.com<br>controlplane.payment-cryptography.eu-west-2.api.aws | HTTPS<br>HTTPS |
| 유럽(파리) | eu-west-3 | controlplane.payment-cryptography.eu-west-3.amazonaws.com<br>controlplane.payment-cryptography.eu-west-3.api.aws | HTTPS<br>HTTPS |

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 |
| --- | --- | --- | --- |
| 남아메리카(상파울루) | sa-east-1 | controlplane.payment-cryptography.sa-east-1.amazonaws.com<br><br>controlplane.payment-cryptography.sa-east-1.api.aws | HTTPS<br><br>HTTPS |

## 데이터 영역 엔드포인트

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 |
| --- | --- | --- | --- |
| 미국 동부(오하이오) | us-east-2 | dataplane.payment-cryptography.us-east-2.amazonaws.com<br><br>dataplane.payment-cryptography.us-east-2.api.aws | HTTPS<br><br>HTTPS |
| 미국 동부(버지니아 북부) | us-east-1 | dataplane.payment-cryptography.us-east-1.amazonaws.com<br><br>dataplane.payment-cryptography.us-east-1.api.aws | HTTPS<br><br>HTTPS |
| 미국 서부(오리곤) | us-west-2 | dataplane.payment-cryptography.us-west-2.amazonaws.com<br><br>dataplane.payment-cryptography.us-west-2.api.aws | HTTPS<br><br>HTTPS |
| 아프리카(케이프타운) | af-south-1 | dataplane.payment-cryptography.af-south-1.amazonaws.com<br><br>dataplane.payment-cryptography.af-south-1.api.aws | HTTPS<br><br>HTTPS |

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 |
| --- | --- | --- | --- |
| 아시아 태평양(하이데라바드) | ap-south-2 | dataplane.payment-cryptography.ap-south-2.amazonaws.com<br>dataplane.payment-cryptography.ap-south-2.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(뭄바이) | ap-south-1 | dataplane.payment-cryptography.ap-south-1.amazonaws.com<br>dataplane.payment-cryptography.ap-south-1.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(오사카) | ap-northeast-3 | dataplane.payment-cryptography.ap-northeast-3.amazonaws.com<br>dataplane.payment-cryptography.ap-northeast-3.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(싱가포르) | ap-southeast-1 | dataplane.payment-cryptography.ap-southeast-1.amazonaws.com<br>dataplane.payment-cryptography.ap-southeast-1.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(시드니) | ap-southeast-2 | dataplane.payment-cryptography.ap-southeast-2.amazonaws.com<br>dataplane.payment-cryptography.ap-southeast-2.api.aws | HTTPS<br>HTTPS |
| 아시아 태평양(도쿄) | ap-northeast-1 | dataplane.payment-cryptography.ap-northeast-1.amazonaws.com<br>dataplane.payment-cryptography.ap-northeast-1.api.aws | HTTPS<br>HTTPS |

| 리전 이름 | 리전 | 엔드포인트 | 프로토콜 |
|---|---|---|---|
| 캐나다(중부) | ca-central-1 | dataplane.payment-cryptography.ca-central-1.amazonaws.com<br>dataplane.payment-cryptography.ca-central-1.api.aws | HTTPS<br>HTTPS |
| 유럽(프랑크푸르트) | eu-central-1 | dataplane.payment-cryptography.eu-central-1.amazonaws.com<br>dataplane.payment-cryptography.eu-central-1.api.aws | HTTPS<br>HTTPS |
| 유럽(아일랜드) | eu-west-1 | dataplane.payment-cryptography.eu-west-1.amazonaws.com<br>dataplane.payment-cryptography.eu-west-1.api.aws | HTTPS<br>HTTPS |
| 유럽(런던) | eu-west-2 | dataplane.payment-cryptography.eu-west-2.amazonaws.com<br>dataplane.payment-cryptography.eu-west-2.api.aws | HTTPS<br>HTTPS |
| 유럽(파리) | eu-west-3 | dataplane.payment-cryptography.eu-west-3.amazonaws.com<br>dataplane.payment-cryptography.eu-west-3.api.aws | HTTPS<br>HTTPS |
| 남아메리카(상파울루) | sa-east-1 | dataplane.payment-cryptography.sa-east-1.amazonaws.com<br>dataplane.payment-cryptography.sa-east-1.api.aws | HTTPS<br>HTTPS |

# AWS Payment Cryptography 시작하기

AWS Payment Cryptography를 시작하려면 먼저 키를 생성한 다음 다양한 암호화 작업에 사용해야 합니다. 아래 자습서에서는 CVV2 값을 생성/확인하는 데 사용할 키를 생성하는 간단한 사용 사례를 제공합니다. 다른 예제를 시도하고 AWS 내에서 배포 패턴을 살펴보려면 다음 AWS Payment Cryptography 워크숍을 시도하거나 GitHub에서 사용할 수 있는 샘플 프로젝트를 살펴보세요.

이 자습서에서는 단일 키를 만들고 키를 사용하여 암호화 작업을 수행하는 방법을 안내합니다. 그런 다음 더 이상 필요하지 않을 경우 키를 삭제하면 키 수명 주기가 완료됩니다.

> **Warning**
>
> 이 사용 설명서의 예제에서는 샘플 값을 사용할 수 있습니다. 키 일련 번호와 같은 프로덕션 환경에서는 샘플 값을 사용하지 않는 것이 좋습니다.

주제


- 사전 조건
- 1단계: 키 생성
- 2단계: 키를 사용하여 CVV2 값 생성
- 3단계: 2단계에서 생성된 값 확인
- 4단계: 네거티브 테스트 수행
- 5단계: (선택 사항) 정리


## 사전 조건

시작하기 전에 다음을 확인하세요.

- 서비스에 액세스할 권한이 있음. 자세한 내용을 알아보려면 IAM 정책을 참조하세요.
- AWS CLI이 설치되어 있음. AWS SDKs 또는 AWS APIs 사용하여 AWS Payment Cryptography에 액세스할 수도 있지만이 자습서의 지침은를 사용합니다 AWS CLI.

# 1단계: 키 생성

첫 번째 단계는 키를 만드는 것입니다. 이 자습서에서는 CVV/CVV2 값을 생성하고 확인하기 위한 CVK 이중 길이 3DES(2KEY TDES) 키를 생성합니다.

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

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
    "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"
    }
}
```

키를 나타내는 KeyArn에 주의하세요(예: arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi). 다음 단계에서 이 작업을 수행합니다.

## 2단계: 키를 사용하여 CVV2 값 생성

이 단계에서는 1단계의 키를 사용하여 지정된 PAN 및 만료일에 대한 CVV2를 생성합니다.

```
$  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"
}
```

cardDataValue(이 경우 3자리 숫자 144)를 기록해 두세요. 다음 단계에서 이 작업을 수행합니다.

## 3단계: 2단계에서 생성된 값 확인

이 예시에서는 1단계에서 생성한 키를 사용하여 2단계의 CVV2를 검증합니다.

다음 명령을 실행하여 CVV2를 검증합니다.

```
$  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"
}
```

이 서비스는 CVV2를 검증했음을 나타내는 HTTP 응답 200을 반환합니다.

## 4단계: 네거티브 테스트 수행

이 단계에서는 CVV2가 올바르지 않고 검증되지 않는 음성 테스트를 생성합니다. 1단계에서 만든 키를 사용하여 잘못된 CVV2을 검증하려고 합니다. 예를 들어 카드 소유자가 결제 시 잘못된 CVV2를 입력한 경우, 이 작업은 필요한 작업입니다.

```
$  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.
```

이 서비스는 “카드 검증 데이터 확인에 실패했습니다”라는 메시지와 INVALID_VALIDATION_DATA라는 사유와 함께 400의 HTTP 응답을 반환합니다.

## 5단계: (선택 사항) 정리

이제 1단계에서 생성한 키를 삭제할 수 있습니다. 복구할 수 없는 변경 사항을 최소화하기 위한 기본 키 삭제 기간은 7일입니다.

```
$ 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"
    }
}
```


출력의 두 필드를 기록해 두세요. deletePendingTimestamp는 기본적으로 향후 7일로 설정되어 있습니다. keyState는 DELETE_PENDING로 설정되어 있습니다. restore-key를 호출하여 예정된 삭제 시간 전에 언제든지 이 삭제를 취소할 수 있습니다.

# 키 관리

AWS Payment Cryptography를 시작하려면 AWS Payment Cryptography 키를 생성합니다.

이 섹션에서는 수명 주기 동안 다양한 AWS Payment Cryptography 키 유형을 생성하고 관리하는 방법을 설명합니다. 키를 생성, 보기 및 편집하는 방법과 키에 태그를 지정하고, 키 별칭을 생성하고, 키를 활성화 또는 비활성화하는 방법을 알아봅니다.

AWS Payment Cryptography 키는 리전 리소스입니다. 여러에서 지정된 키를 사용하려는 경우 동일한 AWS 파티션 및 계정 내에서 지정한 키 구성 요소와 메타데이터를 안전하게 복사하는 다중 리전 키 복제를 활성화할 AWS 리전수 AWS 리전 있습니다. 다중 리전 키 복제의 소스 키를 프라이머리 리전 키(PRK)라고 하며, 이는 모든 키 관리 활동의 신뢰할 수 있는 소스로 남아 있습니다. 복제된 키를 복제본 리전 키(RRK)라고 하며 PRK의 읽기 전용 복제본입니다. 가용성, 재해 복구 및 짧은 지연 시간에 대한 설계 목표를 달성하려면 키와 함께 다중 리전 키를 사용하는 것이 좋습니다.

주제

- 키 생성
- 키 나열
- 키 활성화 및 비활성화
- AWS Payment Cryptography 키 복제
- 키 삭제
- 키 가져오기 및 내보내기
- 별칭 사용
- 키 가져오기
- 키 태그 지정
- AWS Payment Cryptography 키의 키 속성 이해

## 키 생성

CreateKey API 작업을 사용하여 AWS Payment Cryptography 키를 생성할 수 있습니다. 키를 생성할 때 키 알고리즘, 키 사용, 허용된 작업, 내보내기 가능 여부와 같은 속성을 지정합니다. AWS Payment Cryptography 키를 생성한 후에는 이러한 속성을 변경할 수 없습니다.

> **Note**
>
> 에 대해 다중 리전 키 복제가 활성화되어 AWS 계정 있고 Payment Cryptography 키를 생성하는 경우이 키는 자동으로 기본 리전 키(PRK)가 됩니다. PRK는 CreateKey 명령에서 `--replication-regions` 파라미터를 지정하지 않은 경우에도 복제됩니다. 자세한 내용은 다중 리전 키 복제 작동 방식 단원을 참조하십시오.

예제

- 3KEY TDES 기본 파생 키 생성
- CVV/CVV2용 2KEY TDES 키 생성
- HMAC 키 생성
- AES-256 키 생성
- PIN 암호화 키(PEK) 생성
- 비대칭(RSA) 키 생성
- PIN 확인 값(PVV) 키 생성
- 비대칭 ECC 키 생성

## 3KEY TDES 기본 파생 키 생성

Example

이 명령은 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에 복제될 3KEY TDES 파생 키를 생성합니다. 응답에는 reques 파라미터, 후속 호출을 위한 Amazon 리소스 이름(ARN), 키 확인 값(KCV)이 포함됩니다.

```
$ aws payment-cryptography create-key --exportable --key-attributes \
     "KeyUsage=TR31_B0_BASE_DERIVATION_KEY, \
     KeyClass=SYMMETRIC_KEY,KeyAlgorithm=TDES_3KEY, \
     KeyModesOfUse={NoRestrictions=true}" \
     --replication-regions us-east-2 --region us-west-2
```

출력 예시:

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

```
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "FE23D3",
        "KeyAttributes": {
            "KeyAlgorithm": "TDES_3KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": true,
                "Encrypt": false,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": true,
                "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"
}
```

# CVV/CVV2용 2KEY TDES 키 생성

Example

이 명령은 CVV/CVV2 값을 생성하고 확인하기 위한 2KEY TDES 키를 생성합니다. 응답에는 요청 파라미터, 후속 호출을 위한 Amazon 리소스 이름(ARN) 및 키 확인 값(KCV)이 포함됩니다.

```
$ 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}'
```

출력 예시:

```
{
    "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"
    }
}
```

# HMAC 키 생성

Example

HMAC 키는 해시 메시지 인증 코드(HMAC)를 생성하거나 확인하는 데 사용됩니다. HMAC 키를 사용하면 키 생성 시 해시 유형(예: HMAC_SHA224 및 HMAC_SHA512)이 할당되며 수정할 수 없습니다.

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

출력 예시:

```
{
 "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"
 }
}
```

# AES-256 키 생성

Example

이 명령은 데이터 암호화 및 복호화를 위한 AES-256 대칭 키를 생성합니다. AES 키는 민감한 데이터에 대한 강력한 암호화를 제공하며 카드 소지자 데이터 및 기타 민감한 정보를 암호화하는 결제 처리에 일반적으로 사용되지만 TDES는 EMV와 같은 발급자 사용 사례에 더 일반적으로 사용됩니다.

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

출력 예시:

```
{
    "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"
    }
}
```

# PIN 암호화 키(PEK) 생성

Example

이 명령은 PIN 값을 암호화하기 위한 3KEY TDES 키를 생성하지만 핀 키는 상호 운용성 요구 사항에 따라 AES일 수도 있습니다. 이 키를 사용하여 트랜잭션에서와 같이 확인 중에 PINs 안전하게 저장하거나 PINs 해독할 수 있습니다. 응답에는 요청 파라미터, 후속 호출을 위한 ARN 및 KCV가 포함됩니다.

```
$ 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}'
```

출력 예시:

```
{
    "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"
    }
}
```

# 비대칭(RSA) 키 생성

Example

이 명령은 새로운 비대칭 RSA 2048비트 키 페어를 생성합니다. 새 프라이빗 키와 일치하는 퍼블릭 키를 생성합니다. getPublicCertificate API를 사용하여 퍼블릭 키를 검색할 수 있습니다.

```
$ 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}'
```

출력 예시:

```
{
    "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"
    }
}
```

# PIN 확인 값(PVV) 키 생성

Example

이 명령은 PVV 값을 생성하기 위한 3KEY TDES 키를 생성합니다. 이 키를 사용하여 이후에 계산된 PVV와 비교할 수 있는 PVV를 생성할 수 있습니다. 응답에는 요청 파라미터, 후속 호출을 위한 ARN 및 KCV가 포함됩니다.

```
$ 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}'
```

출력 예시:

```
{
    "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"
    }
}
```

# 비대칭 ECC 키 생성

Example

이 명령은 두 당사자 간에 ECDH(Elliptic Curve Diffie-Hellman) 키 계약을 설정하기 위한 ECC 키 페어를 생성합니다. ECDH를 사용하면 각 당사자는 키 목적 K3 및 사용 모드 X를 사용하여 자체 ECC 키 페어를 생성하고 퍼블릭 키를 교환합니다. 그런 다음 양 당사자는 프라이빗 키와 수신된 퍼블릭 키를 사용하여 공유 파생 키를 설정합니다.

결제 시 암호화 키의 일회용 원칙을 유지하려면 ECDH 키 추출 및 서명과 같은 여러 용도로 ECC 키 페어를 재사용하지 않는 것이 좋습니다.

```
$ 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}'
```

출력 예시:

```
{
    "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"
    }
}
```

# 키 나열

ListKeys 작업을 사용하여 계정 및 리전에서 액세스할 수 있는 키 목록을 가져옵니다.

Example

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

출력 예시:

```
{
    "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"
        }
    ]
}
```

# 키 활성화 및 비활성화

AWS Payment Cryptography 키를 비활성화했다가 다시 활성화할 수 있습니다. 키를 생성하면 필터가 기본적으로 활성화됩니다. 키를 비활성화하면 다시 활성화할 때까지 어떠한 암호화 작업에서도 사용할 수 없습니다. 사용 시작/중지 명령은 즉시 적용되므로 변경하기 전에 사용량을 검토하는 것이 좋습니다. 선택적 `timestamp` 파라미터를 사용하여 향후에 적용할 변경 사항(사용 시작 또는 중지)을 설정할 수도 있습니다.

일시적이고 쉽게 취소할 수 있으므로 AWS Payment Cryptography 키를 비활성화하는 것은 파괴적이고 되돌릴 수 없는 작업인 AWS Payment Cryptography 키를 삭제하는 것보다 더 안전한 대안입니다. AWS Payment Cryptography 키 삭제를 고려하는 경우 먼저 비활성화하고 나중에 키를 사용하여 데이터를 암호화하거나 복호화할 필요가 없도록 합니다.

주제

- 키 사용 시작
- 키 사용 중지

## 키 사용 시작

암호화 작업에 키를 사용하려면 키 사용을 활성화해야 합니다. 키가 활성화되지 않은 경우 이 작업을 사용하여 키를 사용할 수 있게 만들 수 있습니다. 이 `UsageStartTimestamp` 필드는 키가 활성화된 시점 또는 활성화될 시점을 나타냅니다. 이는 활성화된 토큰의 경우 과거이며, 활성화가 보류 중인 경우 미래입니다.

Example

이 예시에서는 키 사용을 위해 키를 활성화하도록 요청합니다. 응답에는 키 정보가 포함되며 활성화 플래그가 true로 전환되었습니다. 이는 리스트 키 응답 개체에도 반영됩니다.

```
$ 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"
  }
}
```

# 키 사용 중지

더 이상 키를 사용하지 않으려는 경우 키 사용을 중지하여 추가 암호화 작업을 방지할 수 있습니다. 이 작업은 영구적이지 않으므로 키 사용 시작하기를 사용하여 되돌릴 수 있습니다. 향후에 키가 비활성화되도록 설정할 수도 있습니다. 이 `UsageStopTimestamp` 필드는 키가 언제 비활성화되었는지 또는 비활성화될 것인지를 나타냅니다.

Example

이 예시에서는 향후에 키 사용을 중지하도록 요청합니다. 실행 후에는 키 사용 시작을 통해 다시 활성화하지 않는 한 이 키를 암호화 작업에 사용할 수 없습니다. 응답에는 키 정보가 포함되고 활성화 플래그는 false로 전환되었습니다. 이는 리스트 키 응답 개체에도 반영됩니다.

```
$ 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"
  }
}
```

# AWS Payment Cryptography 키 복제

AWS Payment Cryptography는 다중 리전 키 복제를 지원하므로 특정 AWS Payment Cryptography 키의 키 구성 요소와 메타데이터를 동일한 AWS 파티션 및 계정 AWS 리전 내의 하나 이상의에 안전하게 배포할 수 있습니다.

소스 키는 프라이머리 리전 키(PRK)라고 하며 모든 키 관리 활동에 대한 신뢰할 수 있는 소스로 남아 있는 반면, PRK 및 복제본 리전 키(RRK)는 각각에서 암호화 작업에 사용할 수 있습니다 AWS 리전.

## 다중 리전 키 복제의 이점

다음은 다중 리전 키 복제의 몇 가지 이점을 간략하게 설명합니다.

- 고가용성 애플리케이션을 더 쉽게 설정 - AWS Payment Cryptography는 키 배포를 처리하므로 특정 키의 분리된 복사본을 생성할 필요 없이 여러 AWS 리전 에서 키를 사용할 수 있습니다.
- 고가용성 및 짧은 지연 시간 키 - 다중 리전 키 복제를 사용하면 여러에서 키에 액세스하여 고가용성을 AWS 리전 확보하여 지연 시간을 줄일 수 있습니다.
- 키 구성 요소 내구성 - 복제본 리전 키는 완전한 키 복제본이며 암호화 작업에서 기본 리전 키와 독립적으로 사용할 수 있습니다. RRK는 PRK의 치명적인 데이터 손실 시 내구성 있는 복제본을 제공합니다.

## 다중 리전 키 복제 작동 방식

다중 리전 키 복제가 활성화되면 AWS Payment Cryptography 서비스는 보안 키 배포 메커니즘을 사용하여 키 구성 요소와 메타데이터를 지정한 복제본에 복사 AWS 리전 합니다. 키 속성, 상태 및 활성화와 같은 기본 리전 키 메타데이터에 대한 변경 사항은 복제본 리전 키에 자동으로 복제됩니다.

## 제한 사항 및 고려 사항

다음은 몇 가지 다중 리전 키 복제 제한 사항 및 고려 사항입니다.

- AWS 리전 또는 특정 Payment Cryptography 키에 대해이 기능을 활성화해야 합니다.
  - 에 대해이 기능을 활성화하면 활성화 후 생성된 AWS 리전모든 AWS Payment Cryptography 키가 지정된에 복제됩니다 AWS 리전. 이 리전에서 생성된 키는 기본 리전 키가 됩니다. 이 리전의 기존 키는 자동으로 복제되지 않습니다. 키 수준에서 내의 기존 키에 대해 다중 리전 키 복제를 활성화할 수 AWS 리전 있습니다.
  - 각 에는 고유한 다중 리전 키 복제 설정이 있을 AWS 리전 수 있습니다.

  - 키의 다중 리전 복제 설정이 AWS 리전 다중 리전 키 복제 설정보다 우선합니다.
- 복제본 리전 키는 다른에 복제하도록 구성할 수 없습니다 AWS 리전.
- 다중 리전 키 복제는 트리플 데이터 암호화 표준(3DES), 고급 암호화 표준(AES) 및 해시 기반 메시지 인증 코드(HMAC)와 같은 대칭 결제 암호화 키에 사용할 수 있습니다.
- 비대칭 Payment Cryptography 키는 다중 리전 키 복제를 지원하지 않습니다.
- 복제본 리전 키는 읽기 전용 키입니다. 기본 리전 키에 대한 모든 변경 사항은 복제본 리전 키에 적용됩니다.
- 기본 리전 키 변경은 결국 복제본 리전 키와 일치합니다.
- Payment Cryptography 키는 동일한 AWS 파티션 및 계정으로만 복제할 수 있습니다.
- 복제본 리전 키는 AWS 계정 레벨 AWS Payment Cryptography 한도에 포함됩니다.
- 기본 리전 키와 복제본 리전 키는 동일한 키 식별자를 사용하므로 IAM 정책에서 동일한 ARN으로 두 키를 모두 참조할 수 있습니다.
- 복제가 성공하려면 복제 AWS 리전 본에 `CreateKey` 권한이 있어야 합니다.

## 다중 리전 키 복제 활성화

AWS Payment Cryptography 키에 대해 다중 리전 키 복제를 활성화하는 방법에는 두 가지가 있습니다.

1. AWS 리전: 다중 리전 키 복제는 활성화될 AWS 리전 때 해당에서 생성된 모든 새 키에 적용됩니다. 이 메서드는 모든 키에 대해 일관된 복제를 제공합니다.
2. 특정 AWS Payment Cryptography 키: 개별 키에 대한 다중 리전 키 복제를 관리하여 보다 세분화된 제어 수준을 제공할 수 있습니다.

다중 리전 키 복제가 활성화되면 Payment Cryptography 키가 지정한 AWS 리전 에 복제됩니다.

> **Important**
>
> 다중 리전 키 복제는 일시 중지할 수 없습니다. 복제가 활성화되면 AWS 리전 지정한에 키가 자동으로 복제됩니다. 특정 AWS 리전 또는 Payment Cryptography 키에 대해 다중 리전 키 복제를 비활성화할 수 있습니다. 복제본 리전 키를 삭제하려면 기본 리전 키에서 복제 리전 AWS 리전 으로를 제거해야 합니다.
> 또는 PRK에서 `StopKeyUsage` API 또는 stop-key-usage CLI 명령을 호출하여 PRK 및 연결된 모든 RRKs. 암호화 작업에서는 이러한 키를 사용할 수 없습니다. `StopKeyUsage` API 또는

stop-key-usage CLI 명령을 사용하면 PRK에 대해 활성화된 진행 중인 다중 리전 키 복제가 중지되지 않습니다.

`GetDefaultKeyReplicationRegions` API 또는 get-default-key-replication-regions CLI 명령을 호출하여 특정 AWS 리전 의 AWS Payment Cryptography 키에 대한 다중 리전 키 복제 설정을 확인할 수 있습니다. 이 API 작업 또는 명령을 호출 AWS 리전 하는의 키는 PRK가 됩니다.

다음 절차에 따라 다중 리전 키 복제를 활성화합니다.

For AWS 리전

- 다음 명령을 사용하여 AWS 리전 지정한에 대해 다중 리전 키 복제를 활성화합니다. 이 예제에서는 미국 동부(오하이오) 및 미국 서부(오레곤)에서 다중 리전 키 복제가 활성화됩니다. 이 명령을 사용하려면 예제 명령의 *#### ## ### ####* 자신의 정보로 바꿉니다.

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

> **Note**
>
> 에 대해 다중 리전 키 복제를 활성화해도 기존 AWS Payment Cryptography 키의 복제 구성 AWS 리전 은 변경되지 않습니다. 키 수준에서 기존 키에 대해이 기능을 활성화할 수 있습니다. 에 대해 다중 리전 키 복제가 활성화된 후 생성된 키만 리전 복제 설정을 AWS 리전 사용합니다.

For specific AWS Payment Cryptography keys

- 다음 명령을 사용하여 특정 Payment Cryptography 키에 대해 다중 리전 키 복제를 활성화합니다. 이 예제에서는 미국 동부(오하이오)에서 다중 리전 키 복제가 활성화됩니다. 이 명령을 사용하려면 예제 명령의 *#### ## ### ####* 자신의 정보로 바꿉니다.

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

또는 키 생성 요청에 복제를 포함하여이 기능이 활성화된 새 Payment Cryptography 키를 생성할 수 AWS 리전 있습니다.

> **Note**
>
> 키 복제 설정이 AWS 리전 복제 설정보다 우선합니다.

## 다중 리전 키 복제 비활성화

다중 리전 키 복제를 비활성화하려면 다중 리전 키 복제가 활성화된 방식에 따라 disable-default-key-replication 또는 remove-key-replication-regions CLI 명령을 호출할 수 있습니다. 다중 리전 키 복제를 비활성화하려면 키의 ARN과 AWS 리전 를 지정해야 합니다.

고려 사항

복제 리전 키 삭제는 최종적으로 일관됩니다.

`GetDefaultKeyReplicationRegions` API 또는 get-default-key-replication-regions CLI 명령을 호출하여 특정 AWS 리전 의 AWS Payment Cryptography 키에 대한 다중 리전 키 복제 설정을 확인할 수 있습니다.

다음 절차에 따라 다중 리전 키 복제를 비활성화합니다.

For AWS 리전

- 다음 명령을 사용하여 지정한에 대해 다중 리전 키 복제를 AWS 리전 비활성화합니다. 이 예제에서는 미국 동부(오하이오)에서 다중 리전 키 복제가 비활성화됩니다. 이 명령을 사용하려면 예제 명령의 *#### ## ### ####* 자신의 정보로 바꿉니다.

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

For specific AWS Payment Cryptography keys

- 다음 명령을 사용하여 특정 Payment Cryptography 키에 대한 다중 리전 키 복제를 비활성화합니다. 이 예제에서는 미국 동부(오하이오)에서 다중 리전 키 복제가 비활성화되어 있습니다. 이 명령을 사용하려면 예제 명령의 *#### ## ### ####* 자신의 정보로 바꿉니다.

```
aws payment-cryptography remove-key-replication-regions \
```

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

## 보안 고려 사항

다음은 Payment Cryptography 키에 다중 리전 키 복제를 사용할 때의 보안 고려 사항입니다. 자세한 내용은 AWS Payment Cryptography의 보안 모범 사례 단원을 참조하십시오.

- 키 구성 요소 공유를 제한합니다.
- IAM 정책을 생성할 때 최소 권한의 보안 주체를 따릅니다.
- 복제본 리전 키는 읽기 전용 키이므로 변경할 수 없습니다.

## 모범 사례

다음은 AWS Payment Cryptography 키와 함께 다중 리전 키 복제를 사용할 때의 몇 가지 모범 사례입니다.

- 지정된에 대한 다중 리전 키 복제 AWS 리전 가 즉시 수행되지 않더라도 애플리케이션이 계속 작동하는지 확인합니다. 다중 리전 키 복제가 완료되는 시점을 알아야 하는 경우 GetKey API 작업을 사용하여를 모니터링할 수 있습니다. 를 사용하여 키 복제 이벤트를 모니터링할 수 있습니다AWS CloudTrail.
- 한 리전에서 다른 리전으로 장애 조치 시 자동 배포 프로세스를 테스트하고 구현 AWS 리전 합니다.

## 가격 책정

AWS Payment Cryptography로 생성한 복제본 리전 키에 대한 요금이 부과됩니다. 이러한 키는 당 요금이 부과됩니다 AWS 리전. 최신 Payment Cryptography 요금 정보는 AWS Payment Cryptography 요금 페이지를 참조하세요.

# 키 삭제

AWS Payment Cryptography 키를 삭제하면 키 구성 요소와 키와 연결된 모든 메타데이터가 삭제되며 AWS Payment Cryptography 외부에서 키 사본을 사용할 수 없는 한 되돌릴 수 없습니다. 키가 삭제된 후에는 해당 키로 암호화된 데이터를 더 이상 해독할 수 없습니다. 즉 데이터를 복구할 수 없게 될 수 있습니다. 키를 더 이상 사용할 필요가 없고 다른 당사자가 이 키를 사용하지 않는다고 확신하는 경우에

만 키를 삭제해야 합니다. 확실하지 않은 경우 키 사용을 삭제하는 대신 중지하는 것이 좋습니다. 나중에 다시 사용해야 하는 경우 비활성화된 키를 다시 활성화할 수 있지만 다른 소스에서 다시 가져올 수 없는 한 삭제된 AWS Payment Cryptography 키를 복구할 수 없습니다.

키를 삭제하기 전에 키가 더 이상 필요하지 않은지 확인해야 합니다. AWS Payment Cryptography는 CVV2와 같은 암호화 작업의 결과를 저장하지 않으며 영구 암호화 구성 요소에 키가 필요한지 확인할 수 없습니다.

AWS 삭제를 명시적으로 예약하고 필수 대기 기간이 만료되지 않는 한 Payment Cryptography는 활성 AWS 계정에 속한 키를 삭제하지 않습니다.

그러나 다음 중 하나 이상의 이유로 AWS Payment Cryptography 키를 삭제하도록 선택할 수 있습니다.

- 더 이상 필요하지 않은 키의 키 수명 주기를 완료하기 위해
- 미사용 AWS Payment Cryptography 키 유지 관리와 관련된 관리 오버헤드를 방지하려면

> **Note**
>
> 를 닫거나 삭제 AWS 계정하면 AWS Payment Cryptography 키에 액세스할 수 없게 됩니다. 계정 해지와 별도로 AWS Payment Cryptography 키 삭제를 예약할 필요가 없습니다.

AWS Payment Cryptography는 Payment Cryptography 키 삭제를 예약할 때와 AWS Payment AWS Cryptography 키가 실제로 삭제될 때 AWS CloudTrail 로그에 항목을 기록합니다.

다중 리전 키 복제를 사용하고 프라이머리 리전 키(PRK)인 Payment Cryptography 키를 삭제하면 복제본 리전 키(RRK)도 자동으로 삭제됩니다. RRK는 PRK처럼 삭제할 수 없습니다. RRK를 삭제하려면 PRK의 복제 리전을 수정해야 합니다.

## 대기 기간에 대해

키 삭제는 되돌릴 수 없으므로 AWS Payment Cryptography에서는 대기 기간을 3~180일로 설정해야 합니다. 기본 대기 기간은 7일입니다.

그러나 실제 대기 기간은 예약한 대기 기간보다 최대 24시간 더 길어질 수 있습니다. AWS Payment Cryptography 키가 삭제될 실제 날짜 및 시간을 가져오려면 GetKey 작업을 사용합니다. 시간대를 기록하세요.

대기 기간 동안 AWS Payment Cryptography 키 상태 및 키 상태는 삭제 보류 중입니다.

> **Note**
>
> 삭제 보류 중인 AWS Payment Cryptography 키는 어떤 암호화 작업에서도 사용할 수 없습니다.

대기 기간이 끝나면 AWS Payment Cryptography는 AWS Payment Cryptography 키, 별칭 및 모든 관련 AWS Payment Cryptography 메타데이터를 삭제합니다.

대기 기간을 사용하여 현재 또는 향후에 AWS Payment Cryptography 키가 필요하지 않은지 확인합니다. 대기 기간 동안 키가 필요한 경우 대기 기간이 종료되기 전에 키 삭제를 취소할 수 있습니다. 대기 기간이 종료된 후에는 키 삭제를 취소할 수 없고 서비스가 키를 삭제합니다.

Example

이 예시에서는 키 삭제를 요청합니다. 기본 키 정보 외에도 두 개의 관련 필드는 키 상태가 DELETE_PENDING으로 변경되었다는 것이고 DeletePendingTimestamp는 키가 현재 삭제될 예정인 시기를 나타냅니다.

```
$ 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

이 예시에서는 보류 중인 삭제가 취소됩니다. 성공적으로 완료되면 이전 일정에 따라 키가 더 이상 삭제되지 않습니다. 응답에는 기본 키 정보가 포함되며, 추가로 두 개의 관련 필드 (KeyState 및 deletePendingTimestamp)가 변경되었습니다. KeyState는 CREATE_COMPLETE 값으로 반환되고 DeletePendingTimestamp는 제거됩니다.

```
$ 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"
    }
}
```

# 키 가져오기 및 내보내기

다른 솔루션에서 AWS Payment Cryptography 키를 가져와 HSMs. 많은 고객이 가져오기 및 내보내기 기능을 사용하여 서비스 공급자와 키를 교환합니다. 규정 준수 및 제어를 유지하는 데 도움이 되는 키 관리에 대한 최신 전자 접근 방식을 사용하도록 AWS Payment Cryptography를 설계했습니다. 종이 기반 키 구성 요소 대신 표준 기반 전자 키 교환을 사용하는 것이 좋습니다. 모든 파트너가 전자 키 교환을 지원할 때까지 종이 키 구성 요소를 계속 처리해야 하는 경우를 사용할 수 있습니다물리적 키 교환.

최소 키 강도와 가져오기 및 내보내기 함수에 미치는 영향

PCI에는 암호화 작업, 키 스토리지 및 키 전송을 위한 특정 최소 키 강도가 필요합니다. 이러한 요구 사항은 PCI 표준이 개정될 때 변경될 수 있습니다. 규칙은 스토리지 또는 전송에 사용되는 래핑 키가 최소한 보호되는 키만큼 강력해야 한다고 지정합니다. 다음 표와 같이 내보내기 중에이 요구 사항을 자동으로 적용하고 키가 더 약한 키로 보호되지 않도록 합니다.

다음 표에는 래핑 키, 보호할 키 및 보호 방법의 지원되는 조합이 나와 있습니다.

| | 래핑 키 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 보호할 키 | TDES | TDES | AES_ | AES_ | AES_ | RSA_ | RSA_ | RSA_ | ECC_ | ECC_ | ECC_ | 참고 |
| TDES_2KEY | TR-3 | TR-3 | TR-3 | TR-3 | TR-3 | TR-3 RSA | TR-3 RSA | TR-3 RSA | ECDH | ECDH | ECDH | |
| TDES_3KEY | "지원되지 않음 | TR-3 | TR-3 | TR-3 | TR-3 | TR-3 RSA | TR-3 RSA | TR-3 RSA | ECDH | ECDH | ECDH | |
| AES_128 | "지원되지 | "지원되지 | TR-3 | TR-3 | TR-3 | "지원되지 | TR-3 RSA | TR-3 RSA | ECDH | ECDH | ECDH | |

| | 래핑 키 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 보호할 키 | TDE | TDE | AES_ | AES_ | AES_ | RSA_ | RSA_ | RSA_ | ECC_ | ECC_ | ECC_ | 참고 |
| | 않음 | 않음 | | | | 않음 | | | | | | |
| AES_192 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | TR-3 | TR-3 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | ECDI | ECDI | |
| AES_256 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | TR-3 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | "지원되지 않음 | ECDI | |

자세한 내용은 PCI HSM 표준의 부록 D - 승인된 알고리즘의 최소 및 동등한 키 크기와 강도를 참조하세요.

KEK(키 암호화 키) 교환

ANSI X9.24 TR-34 표준을 사용하는 것이 좋습니다. 이 초기 키 유형을 키 암호화 키(KEK), 영역 마스터 키(ZMK) 또는 영역 제어 마스터 키(ZCMK)라고 할 수 있습니다. 시스템 또는 파트너가 아직 TR-34를 지원하지 않는 경우 RSA 래핑/언래핑을 사용할 수 있습니다. AES-256 키 교환이 필요한 경우 ECDH를 사용할 수 있습니다.

> **Note**
>
> 자체 테스트 키를 가져오거나 키를 기존 HSMs과 동기화하려면 GitHub의 AWS Payment Cryptography 샘플 코드를 참조하세요.

WK (Working Key) 교환

작업 키 교환에는 업계 표준(ANSI X9.24 TR 31-2018 및 X9.143)을 사용합니다. 이를 위해서는 이미 TR-34, RSA Wrap, ECDH 또는 유사한 체계를 사용하여 KEK를 교환해야 합니다. 이 접근 방식은 키 구성 요소를 유형 및 사용량에 항상 암호화 방식으로 바인딩하기 위한 PCI PIN 요구 사항을 충족합니다. 작업 키에는 획득자 작업 키, 발급자 작업 키, BDK 및 IPEK가 포함됩니다.

주제

- 키 가져오기
- 키 내보내기
- 고급 주제

# 키 가져오기

> **Important**
>
> 예를 들어 최신 버전의 AWS CLI V2가 필요합니다. 시작하기 전에 최신 버전으로 업그레이드했는지 확인합니다.

목차


- 키 가져오기 소개
- 대칭 키 가져오기
  - 비대칭 기법을 사용한 키 가져오기(TR-34)
  - 비대칭 기법(ECDH)을 사용하여 키 가져오기
  - 비대칭 기법을 사용하여 키 가져오기(RSA 언래핑)
  - 사전 설정된 키 교환 키(TR-31)를 사용하여 대칭 키를 가져옵니다.
- 비대칭(RSA, ECC) 퍼블릭 키 가져오기
  - RSA 퍼블릭 키 가져오기
  - ECC 퍼블릭 키 가져오기

# 키 가져오기 소개

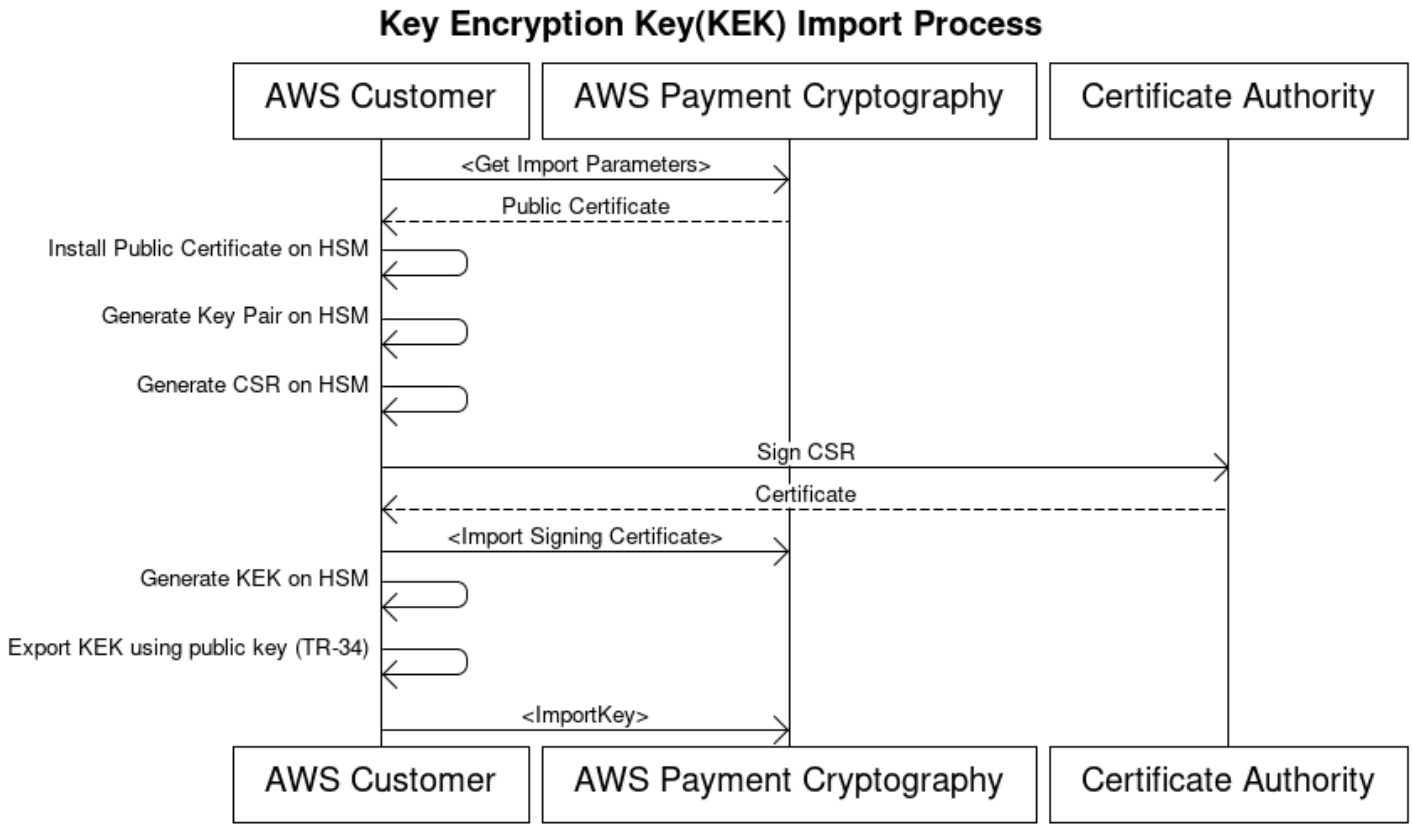
> **Note**
>
> X9.143, TR-31 또는 TR-34 키 블록을 사용하여 키를 가져올 때 AWS Payment Cryptography는 일반적으로 선택적 헤더를 보존(사용하지 않음)합니다. HM(HMAC 해시 유형) 헤더는 암호화 작업 중에 사용됩니다. KP 헤더(래핑 키의 KCV)는 가져오기 프로세스에 고유하며 유지되지 않습니다.

상대방과 키를 교환할 때는 일반적으로 먼저 키 교환 키(KEK)를 교환합니다. 그러면이 키가 후속 키를 보호하는 데 사용됩니다. 전자 형식을 사용하여 KEK는 TR-34, ECDH 또는 RSA 랩과 같은 비대칭 기술을 사용하여 교환할 수 있습니다. 후속 키는 TR-31과 같은 대칭 키 교환을 사용하여 교환됩니다. 이 KEK는 수명이 길며 정책 및 정의된 암호화 기간에 따라 몇 년마다만 업데이트할 수 있습니다.

하나 또는 두 개의 키만 교환하는 경우 비대칭 기법을 사용하여 BDK와 같은 해당 키를 직접 교환하도록 선택할 수도 있습니다. AWS Payment Cryptography는 두 가지 키 교환 방법을 모두 지원합니다.

## 대칭 키 가져오기

비대칭 기법을 사용한 키 가져오기(TR-34)




TR-34는 RSA 비대칭 암호화를 사용하여 교환을 위해 대칭 키를 암호화하고 서명합니다. 이렇게 하면 래핑된 키의 기밀성(암호화)과 무결성(서명)이 모두 보장됩니다.

자체 키를 가져오려면 GitHub에서 AWS Payment Cryptography 샘플 프로젝트를 확인하세요. 다른 플랫폼에서 키를 가져오거나 내보내는 방법에 대한 지침은 GitHub에서 샘플 코드를 사용하거나 해당 플랫폼의 사용 설명서를 참조하세요.

1. 가져오기 초기화 명령을 호출합니다.

   `get-parameters-for-import`를 호출하여 가져오기 프로세스를 초기화합니다. 이 API는 키 가져오기를 위한 키 페어를 생성하고 키에 서명한 다음 인증서와 인증서 루트를 반환합니다. 이 키를 사용하여 내보낼 키를 암호화합니다. TR-34 용어로는 이를 KRD 인증서라고 합니다. 이러한 인증서는 base64로 인코딩되고 수명이 짧으며이 용도로만 사용됩니다. `ImportToken` 값을 저장합니다.

   ```
   $ 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. 키 소스 시스템에 퍼블릭 인증서 설치

   대부분의 HSMs에서는 1단계에서 생성된 퍼블릭 인증서를 설치, 로드 또는 신뢰하여 이를 사용하여 키를 내보내야 합니다. 여기에는 HSM에 따라 전체 인증서 체인 또는 1단계의 루트 인증서만 포함될 수 있습니다.

3. 소스 시스템에서 키 페어를 생성하고 AWS Payment Cryptography에 인증서 체인 제공

   전송된 페이로드의 무결성을 보장하기 위해 전송 당사자(키 배포 호스트 또는 KDH)가 페이로드에 서명합니다. 이를 위해 퍼블릭 키를 생성하고 퍼블릭 키 인증서(X509)를 생성하여 AWS Payment Cryptography에 다시 제공합니다.

   HSM에서 키를 전송할 때 해당 HSM에 키 페어를 생성합니다. HSM, 타사 또는와 같은 서비스가 인증서를 생성할 AWS Private CA 수 있습니다.

   KeyMaterialType이 `RootCertificatePublicKey`이고 KeyUsageType이 인 `importKey` 명령을 사용하여 루트 인증서를 AWS Payment Cryptography에 로드합니다`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`.

   중간 인증서의 경우 `importKey` 명령을의 KeyMaterialType `TrustedCertificatePublicKey` 및의 KeyUsageType과 함께 사용합니다`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`. 여러 중간 인증서에 대해이 프로세스를 반복합니다. 체인에서 마지막으로 가져온 인증서`KeyArn`의를 후속 가져오기 명령에 대한 입력으로 사용합니다.

   > **Note**
   >
   > 리프 인증서를 가져오지 마십시오. 가져오기 명령 중에 직접 제공합니다.

4. 소스 시스템에서 키 내보내기

많은 HSMs 및 관련 시스템은 TR-34 표준을 사용하여 키 내보내기를 지원합니다. 1단계의 퍼블릭 키를 KRD(암호화) 인증서로 지정하고 3단계의 키를 KDH(서명) 인증서로 지정합니다. AWS Payment Cryptography로 가져오려면 형식을 TR-34.2012 비CMS 투 패스 형식으로 지정합니다. 이 형식을 TR-34 Diebold 형식이라고도 합니다.

5. 호출 가져오기 키

KeyMaterialType이 인 importKey API를 호출합니다TR34_KEY_BLOCK. 에 대해 3단계에서 가져온 마지막 CA의 keyARN`certificate-authority-public-key-identifier`,에 대해 4단계의 래핑된 키 구성 요소`key-material`,에 대해 3단계의 리프 인증서를 사용합니다`signing-key-certificate`. 1단계의 import-token을 포함합니다.

```
$ 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.
 \
    }'
```

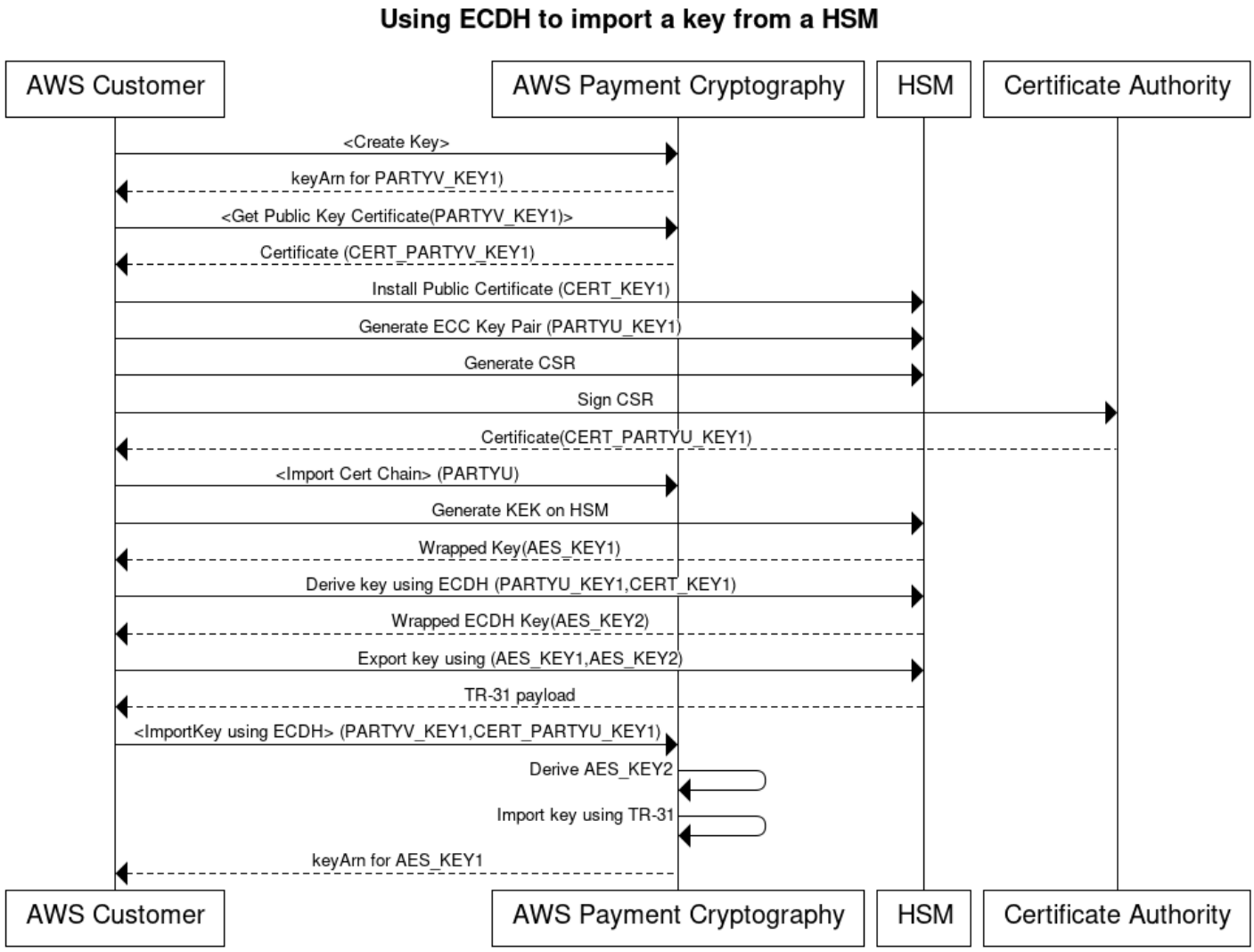
```
{
  "Key": {
    "CreateTimestamp": "2023-06-13T16:52:52.859000-04:00",
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
    "KeyAttributes": {
      "KeyAlgorithm": "TDES_3KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyModesOfUse": {
        "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. 암호화 작업 또는 후속 가져오기에 가져온 키 사용

   가져온 KeyUsage가 TR31_K0_KEY_ENCRYPTION_KEY인 경우 TR-31을 사용하여 후속 키 가져오기에이 키를 사용할 수 있습니다. 다른 키 유형(예: TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY)의 경우 키를 암호화 작업에 직접 사용할 수 있습니다.

## 비대칭 기법(ECDH)을 사용하여 키 가져오기




타원 곡선 Diffie-Hellman(ECDH)은 ECC 비대칭 암호화를 사용하여 사전 교환된 키 없이 두 당사자 간에 공유 키를 설정합니다. ECDH 키는 임시 키이므로 AWS Payment Cryptography는 키를 저장하지 않습니다. 이 프로세스에서 일회성 KBPK/KEK는 ECDH를 사용하여 파생됩니다. 이 파생 키는 전송하려는 실제 키를 래핑하는 데 즉시 사용되며, 이는 다른 KBPK, IPEK 키 또는 기타 키 유형일 수 있습니다.

가져올 때 전송 시스템을 일반적으로 당사자 U(이니시에이터)라고 하고 AWS Payment Cryptography를 당사자 V(응답자)라고 합니다.

> **Note**
>
> ECDH를 사용하여 모든 대칭 키 유형을 교환할 수 있지만 AES-256 키를 안전하게 전송할 수 있는 유일한 접근 방식입니다.

1. ECC 키 페어 생성

   를 호출`create-key`하여이 프로세스에 대한 ECC 키 페어를 생성합니다. 이 API는 키 가져오기 또는 내보내기를 위한 키 페어를 생성합니다. 생성 시이 ECC 키를 사용하여 파생할 수 있는 키 종류를 지정합니다. ECDH를 사용하여 다른 키를 교환(래핑)할 때는 값을 사용합니다`TR31_K1_KEY_BLOCK_PROTECTION_KEY`.

   > **Note**
   >
   > 하위 수준 ECDH는 어떤 목적으로든 사용할 수 있는 파생 키를 생성하지만 AWS Payment Cryptography는 키를 단일 파생 키 유형에만 사용할 수 있도록 허용하여 여러 용도로 키를 실수로 재사용하는 것을 제한합니다.

```
$ 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. 퍼블릭 키 인증서 가져오기

   를 호출`get-public-key-certificate`하여 특정 리전의 AWS Payment Cryptography 전용 계정 CA에서 서명한 X.509 인증서로 퍼블릭 키를 수신합니다.

   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. 상대방 시스템에 퍼블릭 인증서 설치(당사자 U)

   HSMs 많으면 1단계에서 생성된 퍼블릭 인증서를 설치, 로드 또는 신뢰하여 이를 사용하여 키를 내보내야 합니다. 여기에는 HSM에 따라 전체 인증서 체인 또는 1단계의 루트 인증서만 포함될 수 있습니다. 자세한 내용은 HSM 설명서를 참조하세요.

4. 소스 시스템에서 ECC 키 페어를 생성하고 AWS Payment Cryptography에 인증서 체인 제공

   ECDH에서 각 당사자는 키 페어를 생성하고 공통 키에 동의합니다. AWS Payment Cryptography에서 키를 추출하려면 상대방의 퍼블릭 키가 X.509 퍼블릭 키 형식으로 필요합니다.

   HSM에서 키를 전송할 때 해당 HSM에 키 페어를 생성합니다. 키 블록HSMs의 경우 키 헤더는와 비슷합니다`D0144K3EX00E0000`. 인증서를 생성할 때 일반적으로 HSM에서 CSR을 생성한 다음 HSM, 타사 또는와 같은 서비스가 인증서를 생성할 AWS Private CA 수 있습니다.

KeyMaterialType이 RootCertificatePublicKey이고 KeyUsageType이 인 importKey 명령을 사용하여 루트 인증서를 AWS Payment Cryptography에 로드합니다TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE.

중간 인증서의 경우 importKey 명령을의 KeyMaterialType TrustedCertificatePublicKey 및의 KeyUsageType과 함께 사용합니다TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE. 여러 중간 인증서에 대해이 프로세스를 반복합니다. 체인에서 마지막으로 가져온 인증서KeyArn의를 후속 가져오기 명령에 대한 입력으로 사용합니다.

> **Note**
>
> 리프 인증서를 가져오지 마십시오. 가져오기 명령 중에 직접 제공합니다.

5. 당사자 U HSM에서 ECDH를 사용하여 일회성 키 생성

   많은 HSMs 및 관련 시스템은 ECDH를 사용하여 키 설정을 지원합니다. 1단계의 퍼블릭 키를 퍼블릭 키로 지정하고 3단계의 키를 프라이빗 키로 지정합니다. 파생 방법과 같은 허용 가능한 옵션은 API 가이드를 참조하세요.

   > **Note**
   >
   > 해시 유형과 같은 파생 파라미터는 양쪽에서 정확히 일치해야 합니다. 그렇지 않으면 다른 키를 생성합니다.

6. 소스 시스템에서 키 내보내기

   마지막으로 표준 TR-31 명령을 사용하여 AWS Payment Cryptography로 전송할 키를 내보냅니다. ECDH 파생 키를 KBPK로 지정합니다. 내보낼 키는 TR-31 유효한 조합이 적용되는 모든 TDES 또는 AES 키일 수 있습니다. 단, 래핑 키가 내보낼 키만큼 강력해야 합니다.

7. 호출 가져오기 키

   KeyMaterialType이 인 import-key API를 호출합니다DiffieHellmanTr31KeyBlock. 에 대해 3단계에서 가져온 마지막 CA의 KeyARNcertificate-authority-public-key-identifier,에 대해 4단계의 래핑된 키 구성 요소key-material,에 대해 3단계의 리프 인증서를 사용합니다public-key-certificate. 1단계의 프라이빗 키 ARN을 포함합니다.

```
$ 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. 암호화 작업 또는 후속 가져오기에 가져온 키 사용

   가져온 KeyUsage가 TR31_K0_KEY_ENCRYPTION_KEY인 경우 TR-31을 사용하여 후속 키 가져오기에이 키를 사용할 수 있습니다. 다른 키 유형(예: TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY)의 경우 키를 암호화 작업에 직접 사용할 수 있습니다.

비대칭 기법을 사용하여 키 가져오기(RSA 언래핑)

개요: AWS Payment Cryptography는 TR-34가 불가능한 경우 키 교환을 위해 RSA 래핑/언래핑을 지원합니다. TR-34와 마찬가지로이 기법은 RSA 비대칭 암호화를 사용하여 교환을 위한 대칭 키를 암호화합니다. 그러나 TR-34와 달리이 메서드에는 전송 당사자가 페이로드에 서명하지 않습니다. 또한이 RSA 래핑 기법은 키 블록을 포함하지 않으므로 전송 중에 키 메타데이터의 무결성을 유지하지 않습니다.

> **Note**
>
> RSA 랩을 사용하여 TDES 및 AES-128 키를 가져오거나 내보낼 수 있습니다.

1. 가져오기 초기화 명령을 호출합니다.

   를 호출get-parameters-for-import하여 KeyMaterialType의 로 가져오기 프로세스를 초기화합니다KEY_CRYPTOGRAM. TDES 키를 교환할 WrappingKeyAlgorithm 때에 RSA_2048를 사용합니다. TDES RSA_3072 또는 AES-128 키를 교환RSA_4096할 때 또는를 사용합니다. 이 API는 키 가져오기를 위한 키 페어를 생성하고, 인증서 루트를 사용하여 키에 서명하고, 인증서 루트와 인증서 루트를 모두 반환합니다. 이 키를 사용하여 내보낼 키를 암호화합니다. 이러한 인증서는 수명이 짧으며이 용도로만 사용됩니다.

```
$ 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. 키 소스 시스템에 퍼블릭 인증서 설치

   HSMs 많은 경우 1단계에서 생성된 퍼블릭 인증서(및/또는 루트)를 설치, 로드 또는 신뢰하여 이를 사용하여 키를 내보내야 합니다.

3. 소스 시스템에서 키 내보내기

   많은 HSMs 및 관련 시스템은 RSA 랩을 사용하여 키 내보내기를 지원합니다. 1단계의 퍼블릭 키를 암호화 인증서(`WrappingKeyCertificate`)로 지정합니다. 신뢰 체인이 필요한 경우 1단계`WrappingKeyCertificateChain`의를 사용합니다. HSM에서 키를 내보낼 때 패딩 모드 = PKCS#1 v2.2 OAEP(SHA 256 또는 SHA 512 사용)를 사용하여 형식을 RSA로 지정합니다.

4. 호출 import-key

   의를 사용하여 import-key API를 호출`KeyMaterialType`합니다`KeyMaterial`. 1단계`ImportToken`의와 3단계의 `key-material` (래핑된 키 구성 요소)가 필요합니다. RSA 랩은 키 블록을 사용하지 않으므로 키 파라미터(예: 키 사용)를 제공합니다.

```
$ 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
```

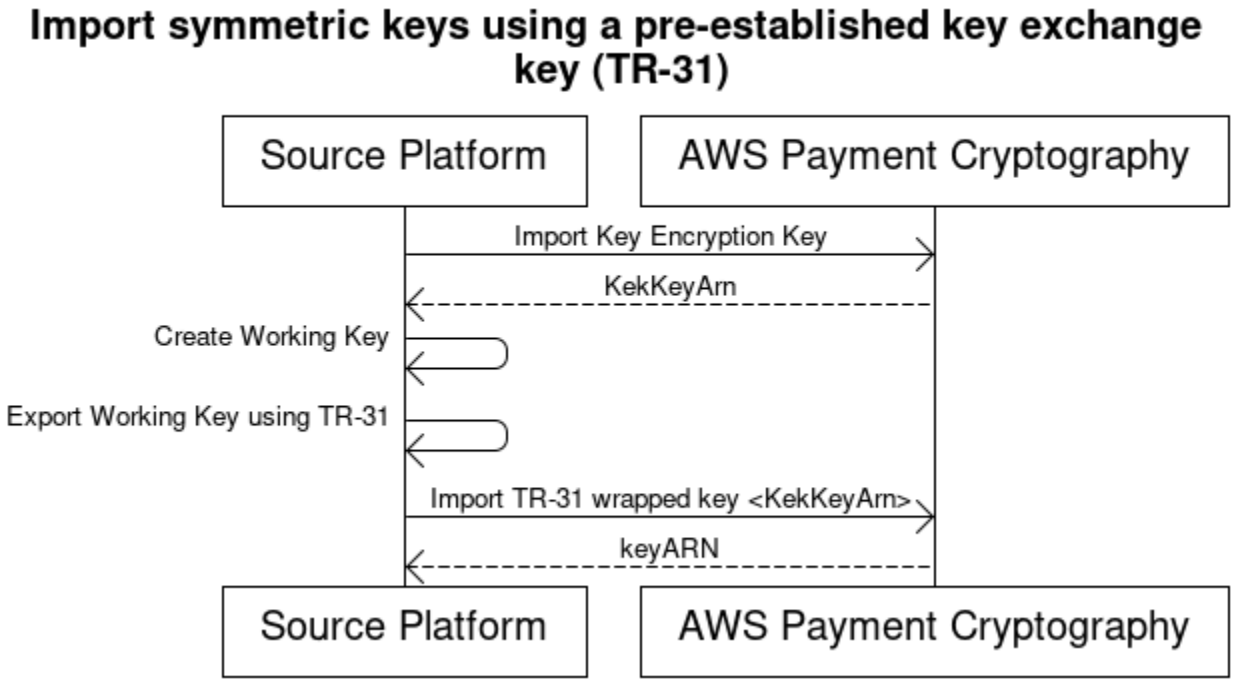
```
{
  "Key": {
    "KeyOrigin": "EXTERNAL",
    "Exportable": true,
    "KeyCheckValue": "DA1ACF",
    "UsageStartTimestamp": 1697643478.92,
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
    "CreateTimestamp": 1697643478.92,
    "KeyState": "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. 암호화 작업 또는 후속 가져오기에 가져온 키 사용

   가져온이 TR31_K0_KEY_ENCRYPTION_KEY 또는 KeyUsage인 경우 TR-31을 사용한 후속 키 가져오기에이 키를 사용할 TR31_K1_KEY_BLOCK_PROTECTION_KEY수 있습니다. 키 유형이 다른 유형(예: TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY)인 경우 키를 암호화 작업에 직접 사용할 수 있습니다.

사전 설정된 키 교환 키(TR-31)를 사용하여 대칭 키를 가져옵니다.




여러 키를 교환하거나 키 교체를 지원할 때 파트너는 일반적으로 먼저 초기 키 암호화 키(KEK)를 교환합니다. TR-34 또는와 같은 기술을 사용하여 AWS Payment Cryptography와 KEK를 교환할 수 있습니다물리적 키 교환.

KEK를 설정한 후 이를 사용하여 후속 키(다른 KEKs. AWS Payment Cryptography는 HSM 공급업체에서 널리 사용되고 지원하는 ANSI TR-31을 사용하여이 키 교환을 지원합니다.

1. 키 암호화 키 가져오기(KEK)

   KEK를 이미 가져왔고 keyARN(또는 keyAlias)을 사용할 수 있는지 확인합니다.

2. 소스 플랫폼에서 키 생성

   키가 없는 경우 소스 플랫폼에서 생성합니다. 또는 AWS Payment Cryptography에서 키를 생성하고 export 명령을 사용할 수 있습니다.

3. 소스 플랫폼에서 키 내보내기

   내보낼 때 내보내기 형식을 TR-31로 지정합니다. 소스 플랫폼은 내보낼 키와 사용할 키 암호화 키를 요청합니다.

4. AWS Payment Cryptography로 가져오기

   import-key 명령을 호출할 때에 대한 키 암호화 키의 keyARN(또는 별칭)을 사용합니다.`WrappingKeyIdentifier`. 에 대한 소스 플랫폼의 출력을 사용합니다.`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"
  }
}
```

# 비대칭(RSA, ECC) 퍼블릭 키 가져오기

가져온 모든 인증서는 최소한 체인의 발급(이전) 인증서만큼 강력해야 합니다. 즉, RSA_2048 CA는 RSA_2048 리프 인증서를 보호하는 데만 사용할 수 있으며 ECC 인증서는 동등한 강도의 다른 ECC 인증서로 보호해야 합니다. ECC P384 인증서는 P384 또는 P521 CA에서만 발급할 수 있습니다. 모든 인증서는 가져올 때 만료되지 않아야 합니다.

RSA 퍼블릭 키 가져오기

AWS Payment Cryptography는 퍼블릭 RSA 키를 X.509 인증서로 가져오는 것을 지원합니다. 인증서를 가져오려면 먼저 루트 인증서를 가져옵니다. 모든 인증서는 가져올 때 만료되지 않아야 합니다. 인증서는 PEM 형식이어야 하며 base64로 인코딩되어야 합니다.

1. 루트 인증서를 AWS Payment Cryptography로 가져오기

   루트 인증서를 가져오려면 다음 명령을 사용합니다.

### Example

2. 퍼블릭 키 인증서를 AWS Payment Cryptography로 가져오기

   이제 퍼블릭 키를 가져올 수 있습니다. TR-34와 ECDH는 런타임 시 리프 인증서를 전달하는 데 의존하므로이 옵션은 다른 시스템의 퍼블릭 키를 사용하여 데이터를 암호화할 때만 사용됩니다. KeyUsage는 TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION으로 설정됩니다.

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"
  }
}
```

## ECC 퍼블릭 키 가져오기

AWS Payment Cryptography는 퍼블릭 ECC 키를 X.509 인증서로 가져오는 것을 지원합니다. 인증서를 가져오려면 먼저 루트 CA 인증서와 중간 인증서를 가져옵니다. 모든 인증서는 가져올 때 만료되지 않아야 합니다. 인증서는 PEM 형식이어야 하며 base64로 인코딩되어야 합니다.

1. AWS Payment Cryptography로 ECC 루트 인증서 가져오기

   루트 인증서를 가져오려면 다음 명령을 사용합니다.

### Example

2. 중간 인증서를 AWS Payment Cryptography로 가져오기

   다음 명령을 사용하여 중간 인증서를 가져옵니다.

### Example

3. 퍼블릭 키 인증서(Leaf)를 AWS Payment Cryptography로 가져오기

   리프 ECC 인증서를 가져올 수 있지만 현재 스토리지 외에는 AWS Payment Cryptography에 정의된 함수가 없습니다. 이는 ECDH 함수를 사용할 때 리프 인증서가 런타임에 전달되기 때문입니다.

# 키 내보내기

목차


- 대칭 키 내보내기
  - 비대칭 기법을 사용하여 키 내보내기(TR-34)
  - 비대칭 기법(ECDH)을 사용하여 키 내보내기
  - 비대칭 기법을 사용하여 키 내보내기(RSA 랩)
  - 사전 설정된 키 교환 키(TR-31)를 사용하여 대칭 키를 내보냅니다.
- DUKPT 초기 키 내보내기(IPEK/IK)
- 내보낼 키 블록 헤더 지정
  - 공통 헤더
- 비대칭(RSA) 키 내보내기


## 대칭 키 내보내기

> **Important**
>
> 시작하기 AWS CLI 전에 최신 버전의가 있는지 확인합니다. 업그레이드하려면 설치를 참조하세요 AWS CLI.

### 비대칭 기법을 사용하여 키 내보내기(TR-34)

TR-34는 RSA 비대칭 암호화를 사용하여 교환을 위해 대칭 키를 암호화하고 서명합니다. 암호화는 기밀성을 보호하는 반면 서명은 무결성을 보장합니다. 키를 내보내면 AWS Payment Cryptography가 키 배포 호스트(KDH) 역할을 하며 대상 시스템이 키 수신 디바이스(KRD)가 됩니다.

> **Note**
>
> HSM이 TR-34 내보내기를 지원하지만 TR-34 가져오기를 지원하지 않는 경우 먼저 TR-34를 사용하여 HSM과 AWS Payment Cryptography 간에 공유 KEK를 설정하는 것이 좋습니다. 그런 다음 TR-31을 사용하여 나머지 키를 전송할 수 있습니다.

1. 내보내기 프로세스 초기화

   를 실행get-parameters-for-export하여 키 내보내기를 위한 키 페어를 생성합니다. 이 키 페어를 사용하여 TR-34 페이로드에 서명합니다. TR-34 용어에서 이는 KDH 서명 인증서입니다. 인증서는 수명이 짧으며에 지정된 기간 동안만 유효합니다`ParametersValidUntilTimestamp`.

   > **Note**
   >
   > 모든 인증서는 base64 인코딩에 있습니다.

   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. AWS Payment Cryptography 인증서를 수신 시스템으로 가져오기

   1단계의 인증서 체인을 수신 시스템으로 가져옵니다.

3. 수신 시스템의 인증서 설정

전송된 페이로드를 보호하기 위해 전송 당사자(KDH)는 페이로드를 암호화합니다. 수신 시스템(일반적으로 HSM 또는 파트너의 HSM)은 퍼블릭 키를 생성하고 X.509 퍼블릭 키 인증서를 생성해야 합니다. AWS Private CA 를 사용하여 인증서를 생성할 수 있지만 모든 인증 기관을 사용할 수 있습니다.

인증서를 받은 후 ImportKey 명령을 사용하여 루트 인증서를 AWS Payment Cryptography로 가져옵니다. 이 경우 `KeyMaterialType`을 `RootCertificatePublicKey`로, `KeyUsageType`를 `TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`으로 설정합니다.

리프 인증서에 서명하는 루트 키이기 `KeyUsageType` 때문에를 `TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`로 사용합니다. 잎 인증서를 AWS Payment Cryptography로 가져올 필요가 없습니다. 잎 인증서를 인라인으로 전달할 수 있습니다.

> **Note**
>
> 이전에 루트 인증서를 가져온 경우이 단계를 건너뜁니다. 중간 인증서의 경우를 사용합니다`TrustedCertificatePublicKey`.

4. 키 내보내기

   가 로 `KeyMaterialType` 설정된 ExportKey API를 호출합니다`TR34_KEY_BLOCK`. 다음을 제공해야 합니다.

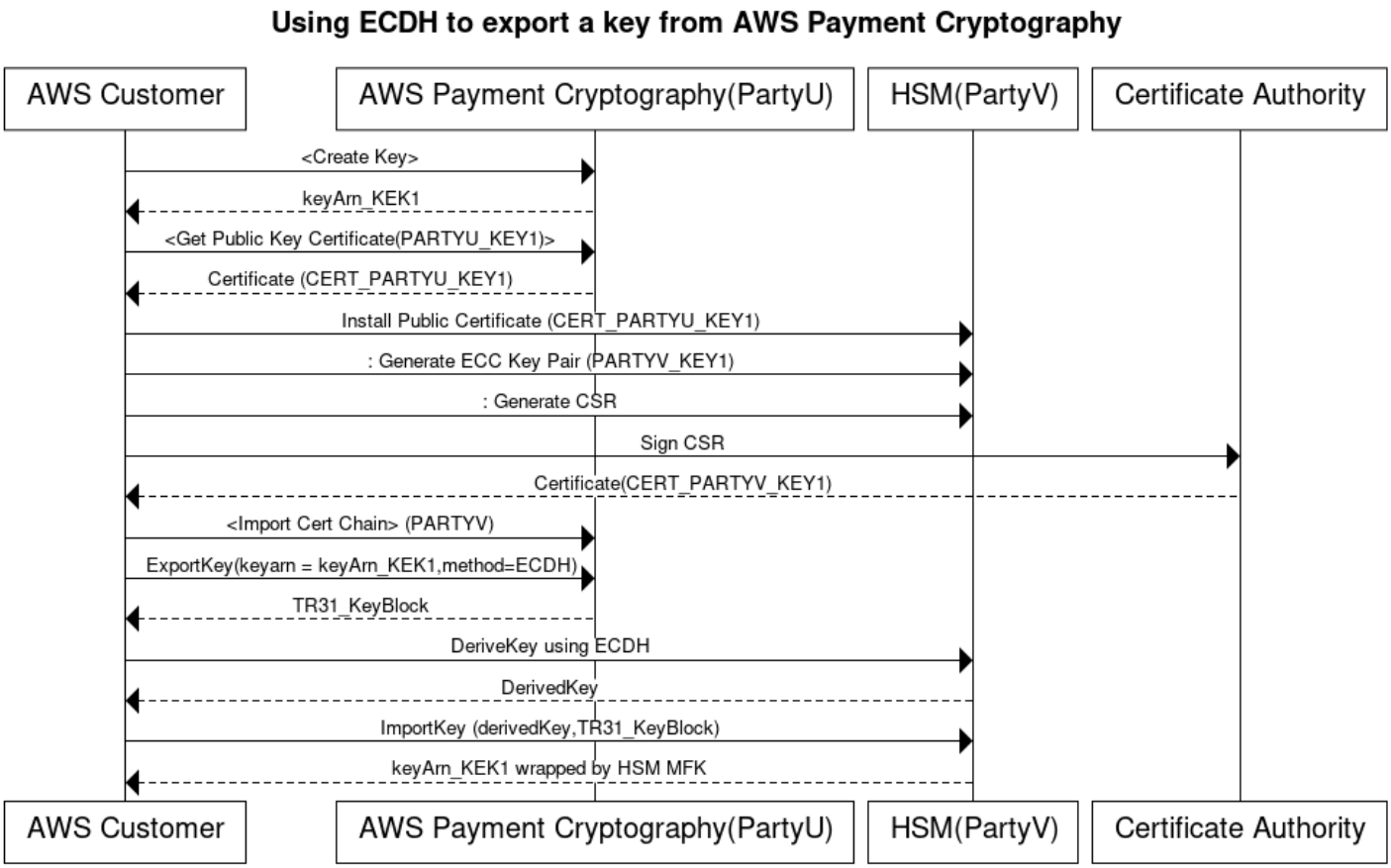
   - 로 3단계에서 루트 CA의 keyARN `CertificateAuthorityPublicKeyIdentifier`
   - 로 3단계의 리프 인증서 `WrappingKeyCertificate`
   - 로 내보내려는 키의 keyARN(또는 별칭) `--export-key-identifier`
   - 1단계의 내보내기 토큰

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"
  }
}
```

## 비대칭 기법(ECDH)을 사용하여 키 내보내기




타원 곡선 Diffie-Hellman(ECDH)은 ECC 비대칭 암호화를 사용하여 사전 교환된 키 없이 두 당사자 간에 공유 키를 설정합니다. ECDH 키는 임시 키이므로 AWS Payment Cryptography는 키를 저장하지 않습니다. 이 프로세스에서 일회성 KBPK/KEK는 ECDH를 사용하여 파생됩니다. 이 파생 키는 전송하려는 키를 래핑하는 데 즉시 사용되며, 이는 다른 KBPK, BDK, IPEK 키 또는 기타 키 유형일 수 있습니다.

내보낼 때 AWS Payment Cryptography를 당사자 U(이니시에이터)라고 하고 수신 시스템을 당사자 V(응답자)라고 합니다.

> **Note**
>
> ECDH는 모든 대칭 키 유형을 교환하는 데 사용할 수 있지만 KEK가 아직 설정되지 않은 경우 AES-256 키를 전송하는 데 사용할 수 있는 유일한 접근 방식입니다.

1. ECC 키 페어 생성

를 호출create-key하여이 프로세스에 대한 ECC 키 페어를 생성합니다. 이 API는 키 가져오기 또는 내보내기를 위한 키 페어를 생성합니다. 생성 시이 ECC 키를 사용하여 파생할 수 있는 키 종류를 지정합니다. ECDH를 사용하여 다른 키를 교환(래핑)할 때는 값을 사용합니다TR31_K1_KEY_BLOCK_PROTECTION_KEY.

> **Note**
>
> 하위 수준 ECDH는 어떤 목적으로든 사용할 수 있는 파생 키를 생성하지만 AWS Payment Cryptography는 키를 단일 파생 키 유형에만 사용할 수 있도록 허용하여 여러 목적으로 키를 실수로 재사용하는 것을 제한합니다.

```
$ 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. 퍼블릭 키 인증서 가져오기

   를 호출`get-public-key-certificate`하여 특정 리전의 AWS Payment Cryptography 전용 계정 CA에서 서명한 X.509 인증서로 퍼블릭 키를 수신합니다.

   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. 상대방 시스템에 퍼블릭 인증서 설치(당사자 V)

   HSMs 많은 경우 키를 설정하려면 1단계에서 생성된 퍼블릭 인증서를 설치, 로드 또는 신뢰해야 합니다. 여기에는 HSM에 따라 전체 인증서 체인 또는 루트 인증서만 포함될 수 있습니다. 구체적인 지침은 HSM 설명서를 참조하세요.

4. 소스 시스템에서 ECC 키 페어를 생성하고 AWS Payment Cryptography에 인증서 체인 제공

   ECDH에서 각 당사자는 키 페어를 생성하고 공통 키에 동의합니다. AWS Payment Cryptography가 키를 추출하려면 상대방의 퍼블릭 키가 X.509 퍼블릭 키 형식으로 필요합니다.

   HSM에서 키를 전송할 때 해당 HSM에 키 페어를 생성합니다. 키 블록HSMs의 경우 키 헤더는와 비슷합니다`D0144K3EX00E0000`. 인증서를 생성할 때 일반적으로 HSM에서 CSR을 생성한 다음 HSM, 타사 또는와 같은 서비스가 인증서를 생성할 AWS Private CA 수 있습니다.

   KeyMaterialTypeKeyUsageType이 인 `importKey` 명령을 사용하여 루트 인증서를 AWS Payment Cryptography에 로드합니다`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`. `RootCertificatePublicKey`

중간 인증서의 경우 `importKey` 명령을의 KeyMaterialType `TrustedCertificatePublicKey` 및의 KeyUsageType과 함께 사용합니다 `TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`. 여러 중간 인증서에 대해이 프로세스를 반복합니다. 체인에서 마지막으로 가져온 인증서`KeyArn`의를 후속 내보내기 명령에 대한 입력으로 사용합니다.

> **Note**
>
> 리프 인증서를 가져오지 마십시오. 내보내기 명령 중에 직접 제공합니다.

5. AWS Payment Cryptography에서 키 추출 및 키 내보내기

내보낼 때 서비스는 ECDH를 사용하여 키를 추출한 다음 즉시 이를 KBPK로 사용하여 TR-31을 사용하여 내보낼 키를 래핑합니다. 내보낼 키는 TR-31 유효한 조합이 적용되는 모든 TDES 또는 AES 키일 수 있습니다. 단, 래핑 키가 내보낼 키만큼 강력해야 합니다.

```
$ 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. 파티 V HSM에서 ECDH를 사용하여 일회성 키 생성

   많은 HSMs 및 관련 시스템은 ECDH를 사용하여 키 설정을 지원합니다. 1단계의 퍼블릭 키를 퍼블릭 키로 지정하고 3단계의 키를 프라이빗 키로 지정합니다. 파생 방법과 같은 허용 가능한 옵션은 API 가이드를 참조하세요.

   > **Note**
   >
   > 해시 유형과 같은 파생 파라미터는 양쪽에서 정확히 일치해야 합니다. 그렇지 않으면 다른 키를 생성합니다.

7. 대상 시스템으로 키 가져오기

   마지막으로 표준 TR-31 명령을 사용하여 AWS Payment Cryptography에서 키를 가져옵니다. ECDH 파생 키를 KBPK로 지정하고 이전에 AWS Payment Cryptography에서 내보낸 TR-31 키 블록을 사용합니다.

### 비대칭 기법을 사용하여 키 내보내기(RSA 랩)

TR-34를 사용할 수 없는 경우 키 교환에 RSA 래핑/언래핑을 사용할 수 있습니다. TR-34와 마찬가지로 이 메서드는 RSA 비대칭 암호화를 사용하여 대칭 키를 암호화합니다. 그러나 RSA 랩에는 다음이 포함되지 않습니다.

- 전송 당사자에 의한 페이로드 서명
- 전송 중에 키 메타데이터 무결성을 유지하는 키 블록

> **Note**
>
> RSA 랩을 사용하여 TDES 및 AES-128 키를 내보낼 수 있습니다.

1. 수신 시스템에서 RSA 키 및 인증서 생성

래핑된 키를 수신하기 위한 RSA 키를 생성하거나 식별합니다. 키는 X.509 인증서 형식이어야 합니다. 인증서를 AWS Payment Cryptography로 가져올 수 있는 루트 인증서로 서명했는지 확인합니다.

2. 루트 퍼블릭 인증서를 AWS Payment Cryptography로 가져오기

   `--key-material` 옵션과 import-key 함께 사용하여 인증서 가져오기

```
$ 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. 키 내보내기

   잎 인증서를 사용하여 키를 내보내도록 AWS Payment Cryptography에 지시합니다. 다음을 지정해야 합니다.

   - 2단계에서 가져온 루트 인증서의 ARN
   - 내보내기를 위한 리프 인증서
   - 내보낼 대칭 키

   출력은 대칭 키의 16진수로 인코딩된 이진 래핑(암호화) 버전입니다.

Example예 - 키 내보내기

```
$ 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. 키를 수신 시스템으로 가져오기

   많은 HSMs 및 관련 시스템은 RSA 언래핑( AWS Payment Cryptography 포함)을 사용하여 키 가져오기를 지원합니다. 가져올 때 다음을 지정합니다.

   - 1단계의 퍼블릭 키를 암호화 인증서로 사용
   - RSA 형식
   - 패딩 모드 - PKCS#1 v2.2 OAEP(SHA 256 사용)

> **Note**
>
> 래핑된 키는 hexBinary 형식으로 출력됩니다. 시스템에 base64와 같은 다른 바이너리 표현이 필요한 경우 형식을 변환해야 할 수 있습니다.

사전 설정된 키 교환 키(TR-31)를 사용하여 대칭 키를 내보냅니다.

여러 키를 교환하거나 키 교체를 지원하는 경우 파트너는 일반적으로 먼저 초기 키 암호화 키(KEK)를 교환합니다. TR-34 또는와 같은 기술을 사용하여 AWS Payment Cryptography와 KEK를 교환할 수 있습니다물리적 키 교환. KEK를 설정한 후 이를 사용하여 다른 KEKs. HSM 공급업체에서 널리 지원하는 ANSI TR-31을 사용하여이 키 교환을 지원합니다.

1. 키 암호화 키(KEK) 설정

   KEK를 이미 교환했고 keyARN(또는 keyAlias)을 사용할 수 있는지 확인합니다.

2. AWS Payment Cryptography에서 키 생성

   키가 아직 없는 경우 키를 생성합니다. 또는 다른 시스템에서 키를 생성하고 가져오기 명령을 사용할 수 있습니다.

3. AWS Payment Cryptography에서 키 내보내기

   TR-31 형식으로 내보낼 때 내보낼 키와 사용할 래핑 키를 지정합니다.

Example예 - TR31 키 블록을 사용하여 키 내보내기

```
$ 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. 키를 시스템으로 가져오기

   시스템의 가져오기 키 구현을 사용하여 키를 가져옵니다.

## DUKPT 초기 키 내보내기(IPEK/IK)

DUKPT를 사용하는 경우 터미널 플릿에 대해 단일 기본 파생 키(BDK)를 생성할 수 있습니다. 터미널은 BDK에 직접 액세스할 수 없습니다. 대신 각 터미널은 IPEK 또는 초기 키(IK)라고 하는 고유한 초기 터미널 키를 받습니다. 각 IPEK는 고유한 키 일련 번호(KSN)를 사용하여 BDK에서 파생됩니다.

KSN 구조는 암호화 유형에 따라 다릅니다.

- TDES의 경우: 10바이트 KSN에는 다음이 포함됩니다.
  - 키 세트 ID의 경우 24비트
  - 터미널 ID의 경우 19비트
  - 트랜잭션 카운터의 경우 21비트
- AES의 경우: 12바이트 KSN에는 다음이 포함됩니다.
  - BDK ID의 경우 32비트
  - 파생 식별자(ID)의 경우 32비트

- 트랜잭션 카운터의 경우 32비트

이러한 초기 키를 생성하고 내보내는 메커니즘을 제공합니다. TR-31, TR-34 또는 RSA 래핑 방법을 사용하여 생성된 키를 내보낼 수 있습니다. IPEK 키는 유지되지 않으며 AWS Payment Cryptography의 후속 작업에 사용할 수 없습니다.

KSN의 처음 두 부분 간에 분할을 적용하지 않습니다. 유도 식별자를 BDK와 함께 저장하려면 AWS 태그를 사용할 수 있습니다.

> **Note**
>
> KSN의 카운터 부분(AES DUKPT의 경우 32비트)은 IPEK/IK 파생에 사용되지 않습니다. 예를 들어 12345678901234560001 및 12345678901234569999의 입력은 동일한 IPEK를 생성합니다.

```
$ 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"
}
}
```

## 내보낼 키 블록 헤더 지정

ASC TR-31 또는 TR-34 형식으로 내보낼 때 키 블록 정보를 수정하거나 추가할 수 있습니다. 다음 표에서는 TR-31 키 블록 형식과 내보내기 중에 수정할 수 있는 요소에 대해 설명합니다.

| 키 블록 속성 | 용도 | 내보내기 중에 수정할 수 있나요? | 참고 |
| --- | --- | --- | --- |
| 버전 ID | 키 구성 요소를 보호하는 데 사용되는 방법을 정의합니다. 표준에는 다음이 포함됩니다.<br>• 버전 A 및 C(키 변형 - 더 이상 사용되지 않음)<br>• 버전 B(TDES를 사용한 파생)<br>• 버전 D(AES를 사용한 키 파생) | 아니요 | TDES 래핑 키에는 버전 B를 사용하고 AES 래핑 키에는 버전 D를 사용합니다. 버전 A 및 C는 가져오기 작업에만 지원됩니다. |
| 키 블록 길이 | 나머지 메시지의 길이를 지정합니다. | 아니요 | 이 값은 자동으로 계산됩니다. 사양에 따라 키 패딩을 추가할 수 있으므로 페이로드를 해독하기 전에 길이가 잘못 표시될 수 있습니다. |
| 키 사용 | 다음과 같이 키에 허용되는 목적을 정의합니다.<br>• C0(카드 확인)<br>• B0(기본 파생 키) | 아니요 | |
| Algorithm | 기본 키의 알고리즘을 지정합니다. 다음을 지원합니다.<br>• T(TDES)<br>• H(HMAC) | 아니요 | 이 값을 있는 그대로 내보냅니다. |

| 키 블록 속성 | 용도 | 내보내기 중에 수정할 수 있나요? | 참고 |
| --- | --- | --- | --- |
| | • A(AES) | | |
| 키 사용 | 다음과 같은 허용되는 작업을 정의합니다.<br>• 생성 및 확인(C)<br>• Encrypt/Decrypt/Wrap/언래핑(B) | 예* | |
| 키 버전 | 키 교체/회전을 위한 버전 번호를 나타냅니다. 지정하지 않으면 기본값은 00입니다. | 예 - 추가 가능 | |
| 키 내보내기 | 키를 내보낼 수 있는지 여부를 제어합니다.<br>• N - 내보내기 불가<br>• E - X9.24에 따라 내보내기(키 블록)<br>• S - 키 블록 또는 키가 아닌 블록 형식으로 내보내기 | 예* | |
| 선택적 키 블록 | 예 - 추가 가능 | 선택적 키 블록은 키에 암호화 방식으로 바인딩된 이름/값 페어입니다. 예: DUKPT 키의 KeySetID. 이름/값 페어 입력을 기반으로 블록 수, 각 블록의 길이 및 패딩 블록(PB)을 자동으로 계산합니다. | |

* 값을 수정할 때 새 값은 AWS Payment Cryptography의 현재 값보다 더 제한적이어야 합니다. 예제:

- 현재 키 사용 모드가 Generate=True,Verify=True인 경우 이를 Generate=True,Verify=False로 변경할 수 있습니다.
- 키가 이미 내보내기 불가능으로 설정된 경우 키를 내보내기 가능으로 변경할 수 없습니다.

키를 내보내면 내보내는 키의 현재 값이 자동으로 적용됩니다. 그러나 수신 시스템으로 전송하기 전에 이러한 값을 수정하거나 추가할 수 있습니다. 다음은 몇 가지 일반적인 시나리오입니다.

- 키를 결제 터미널로 내보낼 때 일반적으로 터미널은 키만 가져오고 내보내서는 안 `Not Exportable`되므로 내보내기 가능성을 로 설정합니다.
- 연결된 키 메타데이터를 수신 시스템에 전달해야 하는 경우 사용자 지정 페이로드를 생성하는 대신 TR-31 선택적 헤더를 사용하여 메타데이터를 키에 암호화 방식으로 바인딩합니다.
- `KeyVersion` 필드를 사용하여 키 버전을 설정하여 키 교체를 추적합니다.

TR-31/X9.143은 공통 헤더를 정의하지만 Payment AWS Cryptography 파라미터를 충족하고 수신 시스템이 이를 수락할 수 있는 한 다른 헤더를 사용할 수 있습니다. 내보내기 중 키 블록 헤더에 대한 자세한 내용은 API 안내서의 키 블록 헤더를 참조하세요.

다음은 다음과 같은 사양으로 BDK 키(예: KIF)를 내보내는 예입니다.

- 키 버전: 02
- KeyExportability: NON_EXPORTABLE
- KeySetID: 00ABCDEFAB(00은 TDES 키를 나타내고 ABCDEFABCD는 초기 키임)

키 사용 모드를 지정하지 않으므로이 키는 arn:aws:payment-cryptography:us-east-2:111122223333:key/5rplquuwozodpwsp(DeriveKey = true)에서 사용 모드를 상속합니다.

> **Note**
>
> 이 예제에서 내보내기 기능을 내보내기 불가능으로 설정하더라도 KIF는 다음을 수행할 수 있습니다.
>
> - DUKPT에 사용되는 IPEK/IK와 같은 파생 키
> - 이러한 파생 키를 내보내 디바이스에를 설치합니다.

이는 표준에서 특별히 허용됩니다.

```
$ 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"
    }
}
```

공통 헤더

X9.143은 일반적인 사용 사례에 대한 특정 헤더를 정의합니다. HM(HMAC 해시) 헤더를 제외하고 AWS Payment Cryptography는 이러한 헤더를 구문 분석하거나 활용하지 않습니다.

| 헤더 이름 | 용도 | 일반적인 검증 | 참고 |
| --- | --- | --- | --- |
| BI | DUKPT의 기본 파생 키 식별자 | 2개의 16진수 문자(TDES의 경우 00, AES의 경우 11)와 TDES KSI의 경우 10개의 16진수 문자 또는 | (BDK ID, AES DUKPT용) 또는 키 세트 식별자(KSI, TDES DUKPT용)를 포함합니다. BDK ID 또는 KSI를 교 |

| 헤더 이름 | 용도 | 일반적인 검증 | 참고 |
| --- | --- | --- | --- |
| | | BDK ID(AES DUKPT)의 경우 8개의 16진수 문자. | 환할 때 사용할 수 있지만 IK 및 KS 블록에 포함된 다른 데이터를 교환할 필요는 없습니다. 일반적으로 BI는 KIF로 전송할 때 사용되는 반면, IK 또는 KS는 터미널 자체에 주입할 때 사용됩니다. |
| HM | HMAC 작업에 대한 해시 유형을 지정합니다. | • 10 – SHA-1<br>• 20 – SHA-224<br>• 21 – SHA-256<br>• 22 – SHA-384<br>• 23 – SHA-512<br>• 24 – SHA-512/224<br>• 25 – SHA-512/256<br>• 30 – SHA3-224<br>• 31 – SHA3-256<br>• 32 – SHA3-384<br>• 33 – SHA3-512<br>• 40 – SHAKE128<br>• 41 - SHAKE256 | 서비스는 내보내기 시이 필드를 자동으로 채우고 가져오기 시 구문 분석합니다. SHAKE128과 같이 서비스에서 지원하지 않는 해시 유형은 가져올 수 있지만 암호화 함수에는 사용할 수 없습니다. |

| 헤더 이름 | 용도 | 일반적인 검증 | 참고 |
| --- | --- | --- | --- |
| IK | AES DUKPT의 초기 키 일련 번호 | 16 16진수 문자 | 이 값은 수신 디바이스에서 초기 DUKPT 키 사용을 인스턴스화하는 데 사용되며 BDK에서 파생된 초기 키를 식별합니다. 이 필드에는 일반적으로 파생 데이터가 포함되지만 카운터는 포함되지 않습니다. TDES DUKPT용 KS를 사용합니다. |
| KS | TDES DUKPT의 초기 키 일련 번호 | 20 16진수 문자 | 이 값은 수신 디바이스에서 초기 DUKPT 키 사용을 인스턴스화하는 데 사용되며 BDK에서 파생된 초기 키를 식별합니다. 이 필드에는 일반적으로 파생 데이터 + 제로화된 카운터 값이 포함됩니다. AES DUKPT에 IK를 사용합니다. |

| 헤더 이름 | 용도 | 일반적인 검증 | 참고 |
| --- | --- | --- | --- |
| KP | 래핑 키의 KCV | 2개의 16진수 문자는 KCV 메서드를 나타냅니다(X9.24 메서드의 경우 00, CMAC 메서드의 경우 01). 그 뒤에는 일반적으로 616진수 문자인 KCV 값이 옵니다. 예를 들어 010FA329는 01(CMAC) 메서드를 사용하여 계산된 0FA329의 KCV를 나타냅니다. | 이 값은 수신 디바이스에서 초기 DUKPT 키 사용을 인스턴스화하는 데 사용되며 BDK에서 파생된 초기 키를 식별합니다. 이 필드에는 일반적으로 파생 데이터 + 제로화된 카운터 값이 포함됩니다. AES DUKPT에 IK를 사용합니다. |
| PB | 패딩 블록 | 임의의 인쇄 가능한 ASCII 문자 | 서비스는 내보내기 시 이 필드를 자동으로 채워 선택적 헤더가 암호화 블록 길이의 배수가 되도록 합니다. |

## 비대칭(RSA) 키 내보내기

인증서 형식으로 퍼블릭 키를 내보내려면 get-public-key-certificate 명령을 사용합니다. 이 명령은 다음을 반환합니다.

- 인증서
- 루트 인증서

두 인증서 모두 base64 인코딩에 있습니다.

> **Note**
>
> 이 작업은 멱등성이 아닙니다. 후속 호출은 동일한 기본 키를 사용하는 경우에도 다른 인증서를 생성할 수 있습니다.

Example

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

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

# 고급 주제

이 섹션에서는 고급 키 교환 시나리오 및 구성을 다룹니다.

주제

- BYOCA(Bring Your Own Certificate Authority)
- 물리적 키 교환

## BYOCA(Bring Your Own Certificate Authority)

기본적으로 서비스 내에서 생성된 비대칭(RSA,ECC) 키에 퍼블릭 키 인증서가 필요한 경우 이러한 인증서는 AWS Payment Cryptography 및 계정 고유 인증 기관(CA)에서 발급합니다. 이렇게 하면 CA를 식별 또는 설정하거나 인증서 서명 요청(CSR)을 관리할 필요 없이 X.509를 간단하게 사용할 수 있습니다.

AWS Payment Cryptography는 정책 또는 규정 준수 이유로 필요한 경우 자체 CA를 사용할 수 있는 기능도 제공합니다.

개요

BYOCA 기능을 사용하면 TR-34 가져오기/내보내기, RSA 언래핑, ECDH 기반 키 전송 등 인증서가 사용되는 모든 곳에서 자체 인증 기관을 사용할 수 있습니다. 이는 조직 전체에서 일관된 인증서 체인을 유지해야 하거나 특정 CA 인증서가 필요한 파트너와 협력할 때 유용합니다. 다음 예제에서는 TR-34 키 내보내기를 사용하는 BYOCA 워크플로를 보여줍니다.

표준 TR-34 내보내기 흐름과 비교한 세 가지 주요 차이점은 다음과 같습니다.

1. 서명 RSA 키는 CreateKey를 사용하여 명시적으로 생성됩니다. 이전에는 GetParametersForExport를 통해 암시적으로 생성되었습니다.
2. 새 API GetCertificateSigningRequest는 외부 CA에서 서명할 수 있는 인증서 서명 요청(CSR)을 생성합니다.
3. ExportKey API는 런타임에 인증서를 제공할 수 있도록 확장됩니다. 이전에는에서 이를 암시적으로 제공했으며`import-token`, 이는 선택적 필드가 되었습니다.

⚠ 중요 고려 사항

- 이 예제에서는 RSA-2048 키를 사용하고 TDES-2KEY 키를 래핑합니다. AES-128을 내보낼 때 모든 키가 RSA-3072 또는 RSA-4096인지 확인합니다.
- 가장 일반적인 오류는 `SigningKeyIdentifier` 및 로 표시되는 키가 일치하지 않는다는 `SigningKeyCertificate` 것입니다.

BYOCA 워크플로

다음 단계는 TR-34 내보내기를 위한 전체 BYOCA 워크플로를 보여줍니다.

단계(Steps)

- 1단계: RSA 키 생성
- 2단계: 인증서 서명 요청 생성
- 3단계: CSR 검토(선택 사항)
- 4단계: 인증 기관을 통해 CSR 서명
- 5단계: CA 인증서 가져오기
- 6단계: KRD 암호화 인증서 가져오기
- 7단계: BYOCA를 사용하여 키 내보내기

1단계: RSA 키 생성

먼저 궁극적으로 KDH 서명 인증서가 될 RSA 키 페어를 생성합니다. 태그를 추가하여 키의 목적을 식별할 수 있습니다.

Example서명을 위한 RSA 키 생성

```
$ 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"
    }
}
```

다음 단계에서 KeyArn 필요하므로를 기록해 둡니다.

2단계: 인증서 서명 요청 생성

GetCertificateSigningRequest API를 사용하여 외부 CA가 서명할 인증서 서명 요청(CSR)을 생성합니다. 출력은 base64로 인코딩된 PEM 파일입니다. 콘텐츠를 base64로 디코딩하고 저장하는 경우 PEM 형식의 유효한 CSR이 있습니다.

Example CSR 생성

```
$ aws payment-cryptography-data 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..."
}
```

`CertificateSigningRequest` 필드에는 서명을 위해 CA에 보낼 base64로 인코딩된 CSR이 포함되어 있습니다.

3단계: CSR 검토(선택 사항)

선택적으로 OpenSSL을 사용하여 CSR 콘텐츠를 검토하고 예상대로 유효한지 확인할 수 있습니다.

Example OpenSSL을 사용하여 CSR 검토

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

4단계: 인증 기관을 통해 CSR 서명

CSR을 생성한 후에는 인증 기관(CA)의 서명을 받아야 합니다. 프로덕션 환경에서는 일반적으로 AWS Private CA 또는 조직의 설정된 CA 인프라를 사용합니다. 테스트 목적으로 OpenSSL을 사용하여 자체 서명된 인증서를 생성할 수 있습니다.

사용 AWS Private CA

를 사용하여 CSR에 서명하려면 AWS Private CA먼저 base64로 인코딩된 CSR을 디코딩하고 파일에 저장한 다음 IssueCertificate API를 사용합니다.

Example를 사용하여 CSR 서명 AWS Private 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"
}
```

그런 다음 서명된 인증서를 검색합니다.

Example서명된 인증서 검색

```
$ 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-----"
}
```

내보내기 단계에서 사용할 인증서 콘텐츠를 저장합니다. `ExportKey` API에 제공할 때 base64 인코딩이 필요합니다.

테스트에 OpenSSL 사용

테스트 목적으로 OpenSSL을 사용하여 자체 서명된 CA를 생성하고 CSR에 서명할 수 있습니다. 먼저 CA 프라이빗 키와 자체 서명된 인증서를 생성합니다.

Example OpenSSL을 사용하여 테스트 CA 생성

```
$ # 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"
```

그런 다음 이전 단계에서 CSR을 디코딩하고 테스트 CA로 서명합니다.

Example OpenSSL을 사용하여 CSR 서명

```
$ # 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
```

이제 서명된 인증서가에 있습니다`signed-cert.pem`. `ExportKey` API에 인증서를 제공할 때이 인증서를 base64로 인코딩해야 합니다.

Example Base64 서명된 인증서 인코딩

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

5단계: CA 인증서 가져오기

임의의 인증서가 사용되지 않도록 하려면 사용 중인 모든 CA를 먼저 신뢰해야 합니다. ImportKey API를 사용하여 외부 CA의 루트 인증서를 가져옵니다. 중간 CA를 사용하는 경우를 `import-key` 다시 호출하고 `TrustedPublicKey` 대신를 지정`RootCertificatePublicKey`하고 루트 CA ARN을 지정합니다.

Example루트 CA 인증서 가져오기

```
$ 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"
    }
}
```

내보내기 단계에서 KeyArn 사용할 CA를 기록해 둡니다.

6단계: KRD 암호화 인증서 가져오기

이 예제에서는 AWS Payment Cryptography로 다시 가져오므로 GetParametersForImport API를 사용하여 KRD 퍼블릭 키 인증서를 수신하도록 서비스를 호출합니다. 실제 시나리오에서는 HSM, ATM, 결제 터미널 또는 결제 터미널 관리 시스템과 같은 다른 시스템에서 이를 제공합니다.

Example가져오기를 위한 파라미터 가져오기

```
$ aws payment-cryptography-data 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"
}
```

7단계: BYOCA를 사용하여 키 내보내기

마지막으로 ExportKey API를 사용하여 자체 CA 서명 인증서와 함께 TR-34를 사용하여 키를 내보냅니다. ExportKey 외부 CA에서 서명한 서명 인증서를 제공합니다.

Example BYOCA를 사용하여 TR-34 내보내기

```
$ aws payment-cryptography-data 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"
    }
}
```

이제 수신 시스템에서 표준 TR-34 가져오기 프로세스를 사용하여 내보낸 키 블록을 가져올 수 있습니다.

추가 참고 사항:

- 이 예제는 AWS CLI를 사용하여 표시됩니다. Java, Python, Go 및 Rust를 포함한 모든 AWS SDKs에서 동일한 기능을 사용할 수 있습니다.
- 자체 서명된 CA로 테스트하는 경우 OpenSSL을 사용하여 테스트 CA를 생성하고 CSR에 서명할 수 있습니다. 프로덕션 환경에서는 조직의 설정된 CA 인프라를 사용합니다.

## 물리적 키 교환

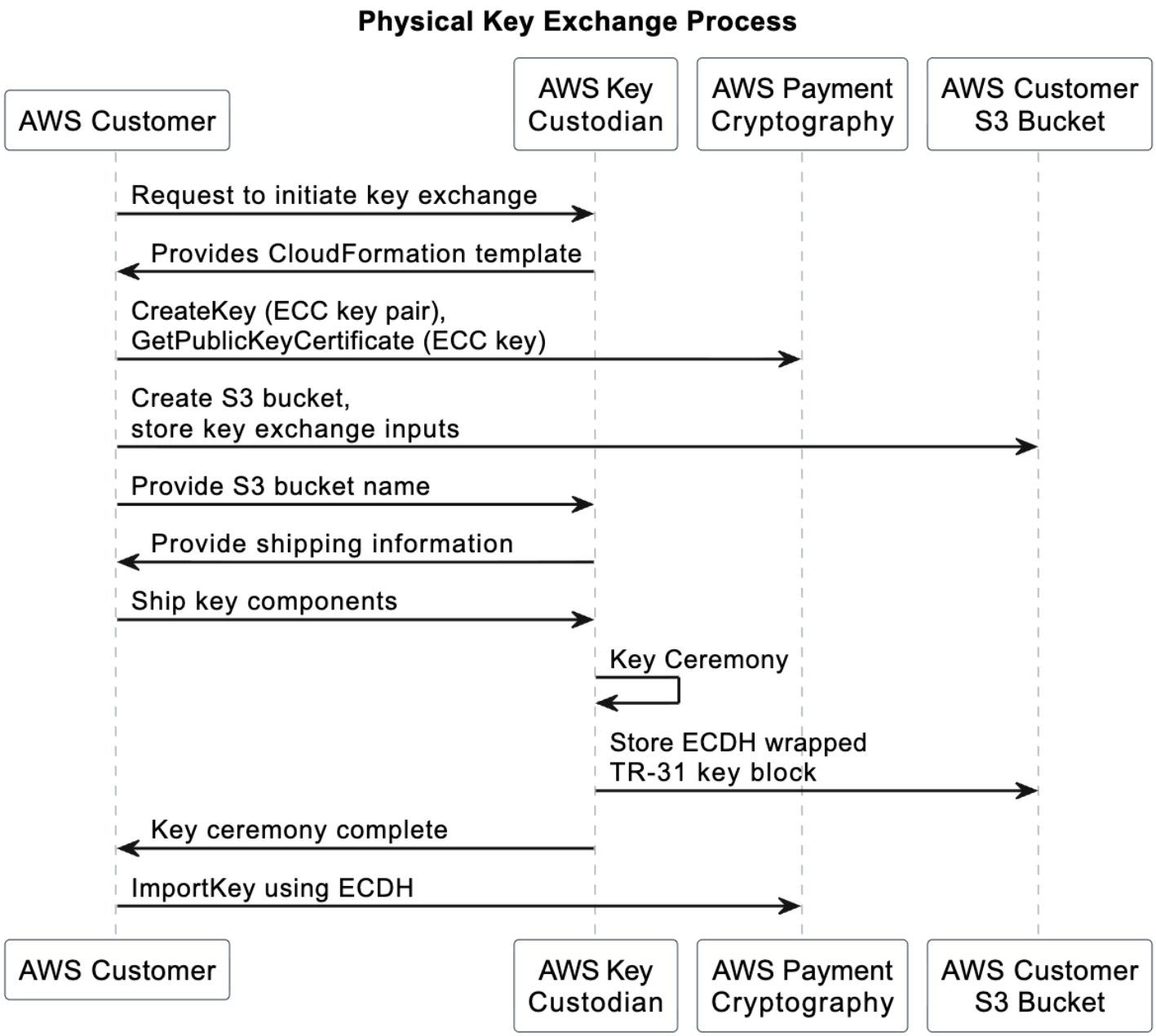
파트너 또는 공급업체가 전자 키 교환을 지원하지 않는 경우 물리적 키 교환을 사용하여 종이 기반 암호화 키 구성 요소를 전자 형식으로 안전하게 변환할 수 있습니다. 훈련된 AWS 키 관리자는 PCI PIN 및 P2PE 인증 AWS 운영 보안 시설에서 키 의식을 수행하여 오프라인 HSM을 사용하여 종이 키 구성 요소를 전자 형식으로 변환합니다. 이 서비스는 ECDH 기반 키 교환을 사용하여 ECDH로 래핑된 TR-31 키 블록을 전송합니다.이 블록은 AWS Payment Cryptography 계정으로 직접 가져옵니다.

> **Note**
>
> 키 가져오기 및 내보내기 가능하면 표준 기반를 사용하는 것이 좋습니다. 파트너 또는 공급업체가 ANSI X9.24 TR-34, RSA 래핑/언래핑 또는 ECDH와 같은 전자 키 교환 방법을 지원하지 않는 경우에만 물리적 키 교환을 사용합니다.

### 물리적 키 교환 작동 방식

종이 키 교환을 시작하기 위해 CloudFormation 템플릿은 계정에 ECC 키 페어와 S3 버킷을 생성하는 등 사전 조건 설정을 안내합니다. 그런 다음 사용자 또는 파트너가 종이 키 구성 요소를 AWS 보안 시설로 배송합니다. 여기서 훈련된 AWS 키 관리인이 오프라인 HSM을 사용하여 키 의식을 수행합니다. 출력은 S3 버킷에 업로드된 ECDH 래핑 TR-31 키 블록으로, 비대칭 기법(ECDH)을 사용하여 키 가져오기 메서드를 사용하여 계정으로 가져옵니다. 물리적 키 교환은 TDES 및 AES 키 알고리즘 모두에서 KEK(키 사용 K1) 또는 BDK(키 사용 B0) 키 가져오기를 지원합니다.

다음 다이어그램은 end-to-end 물리적 키 교환 프로세스를 보여줍니다.




1. 시작 - 지원 티켓을 제출하거나 계정 관리자와 협력하여 요청을 제출합니다.
2. 고객 설정 - AWS 결제 암호화는 다음 사전 조건 단계를 완료할 수 있는 CloudFormation 템플릿을 제공합니다.
   - AWS Payment Cryptography 계정 내에 ECC P521 키 페어를 생성하고 퍼블릭 키 인증서를 검색합니다.
   - AWS Payment Cryptography 서비스 보안 주체에게 읽기/쓰기 액세스 권한을 부여하는 정책을 사용하여 Amazon S3 버킷을 생성합니다.

- ECC 퍼블릭 인증서와 서명 루트 CA를 Amazon S3 버킷에 저장합니다.
- 키 사용, 키 사용 모드, 전송할 종이 키 구성 요소 수와 같은 주요 속성을 제공합니다.

3. S3 버킷 이름 공유 - 고객은 CloudFormation 스택에서 생성한 S3 버킷 이름을 공유합니다. 여기서 퍼블릭 키 인증서, 인증서 체인 및 키 속성은 AWS Payment Cryptography에 저장되어 키 교환을 시작합니다.
4. 배송 조정 - AWS Payment Cryptography는 미국 기반 보안 시설에 대한 배송 세부 정보를 제공합니다. 사용자 또는 파트너가 종이 키 구성 요소를 AWS 키 관리자에게 배송합니다.
5. 구성 요소 수신 - AWS 키 관리자는 각 종이 구성 요소를 수신하고 각 구성 요소에 대해 별도의 승인을 보냅니다.
6. 키 세레모니 - AWS 키 관리인은 오프라인 HSM을 사용하여 키 세레모니를 수행합니다. ECDH에서 파생된 AES-256 키, 오프라인 HSM의 ECC 퍼블릭 인증서 및 서명 인증서를 사용하여 래핑된 결과 TR-31 키 블록은 Amazon S3 버킷에 업로드됩니다.
7. 완료 - AWS 결제 암호화는 키 의식이 완료되었다는 확인을 보냅니다. 그런 다음 비대칭 기법(ECDH)을 사용하여 키 가져오기 메서드를 사용하여 ECDH 래핑 TR-31 키 블록을 AWS Payment Cryptography 계정으로 가져올 수 있습니다.
8. 결제 - 키 행사가 성공적으로 완료되면 교환되는 키당 요금이 청구됩니다.

## 보안 및 규정 준수

물리적 키 교환은 PCI PIN 및 PCI P2PE 물리적 및 논리적 보안 요구 사항을 충족하도록 설계된 AWS 보안 시설에서 작동합니다. 다음과 같은 제어가 마련되어 있습니다.

### 이중 제어 및 업무 분리

AWS 키 관리인은 별도의 보고 구조를 가진 여러 팀에서 할당됩니다. 이중 제어 하에서 키 세리모니 단계가 수행되도록 하는 프로세스가 마련되어 있습니다.

### 오프라인 HSM

주요 의식은 네트워크 연결 없이 오프라인으로 작동하는 인증된 PCI PTS HSM 등록 하드웨어 보안 모듈을 사용하여 수행됩니다. 키는 HSM 경계 외부의 일반 텍스트에 존재하지 않습니다.

### 암호화 키 전송

키 구성 요소는 ECDH 기반 키 교환을 사용하여 오프라인 HSM에서 AWS Payment Cryptography 계정으로 전송되어 end-to-end 암호화 보호를 보장합니다.

감사 및 규정 준수

AWS 에는 PCI PIN 및 P2PE 증명에 대해 정기적으로 평가되는 해당 규정 준수 요구 사항을 충족하는 프로세스가 있습니다. 아 AWS 티팩트의 규정 준수 패키지를 검토하여 자체 PCI 평가에서 참조한 보고서를 확인합니다.

# 별칭 사용

별칭은 AWS Payment Cryptography 키의 표시 이름입니다. 예를 들어 별칭을 사용하면 키를 `arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h` 대신 `alias/test-key`로 지칭할 수 있습니다.

별칭을 사용하여 대부분의 키 관리(컨트롤 플레인) 작업과 암호화(데이터 플레인) 작업에서 키를 식별할 수 있습니다.

정책을 편집하거나 권한 부여를 관리하지 않고도 별칭을 기반으로 AWS Payment Cryptography 키에 대한 액세스를 허용하고 거부할 수도 있습니다. 이 기능은 ABAC(속성 기반 액세스 제어)에 대한 서비스 지원의 일부입니다.

별칭의 강력한 기능은 언제든지 별칭과 연결된 키를 변경할 수 있는 기능에서 비롯됩니다. 별칭을 사용하면 코드를 더 쉽게 작성하고 유지 관리할 수 있습니다. 예를 들어 별칭을 사용하여 특정 AWS Payment Cryptography 키를 참조하고 AWS Payment Cryptography 키를 변경한다고 가정해 보겠습니다. 이 경우 별칭을 다른 키와 연결하기만 하면 됩니다. 코드나 애플리케이션 구성을 변경할 필요가 없습니다.

또한 별칭을 사용하면 다른 AWS 리전에서 동일한 코드를 더 쉽게 재사용할 수 있습니다. 여러 리전에서 동일한 이름의 별칭을 생성하고 각 별칭을 해당 리전의 AWS Payment Cryptography 키와 연결합니다. 코드가 각 리전에서 실행되면 별칭은 해당 리전에서 연결된 AWS Payment Cryptography 키를 참조합니다.

`CreateAlias` API를 사용하여 AWS Payment Cryptography 키의 별칭을 생성할 수 있습니다.

AWS Payment Cryptography API는 각 계정 및 리전의 별칭을 완벽하게 제어합니다. API에는 별칭 생성(CreateAlias), 별칭 이름 및 연결된 keyARN 보기(list-aliases), 별칭과 연결된 AWS Payment Cryptography 키 변경(update-alias), 별칭 삭제(delete-alias) 작업이 포함되어 있습니다.

주제

- 별칭 정보
- 애플리케이션에서 별칭 사용

- 관련 API

## 별칭 정보

AWS Payment Cryptography에서 별칭이 작동하는 방법을 알아봅니다.

별칭은 독립적인 AWS 리소스입니다.

별칭은 AWS Payment Cryptography 키의 속성이 아닙니다. 별칭에 대해 수행하는 작업은 연결된 키에 영향을 주지 않습니다. AWS Payment Cryptography 키의 별칭을 생성한 다음 다른 AWS Payment Cryptography 키와 연결되도록 별칭을 업데이트할 수 있습니다. 연결된 AWS Payment Cryptography 키에 영향을 주지 않고 별칭을 삭제할 수도 있습니다. AWS Payment Cryptography 키를 삭제하면 해당 키와 연결된 모든 별칭이 할당되지 않습니다.

IAM 정책에서 별칭을 리소스로 지정하는 경우 정책은 연결된 AWS Payment Cryptography 키가 아닌 별칭을 참조합니다.

각 별칭은 친숙한 이름으로 합니다.

별칭을 만들 때는 접두사 `alias/`를 앞에 붙인 별칭 이름을 지정합니다. 예를 들면 `alias/test_1234`과 같습니다.

각 별칭은 한 번에 하나의 AWS Payment Cryptography 키와 연결됩니다.

별칭과 해당 AWS Payment Cryptography 키는 동일한 계정 및 리전에 있어야 합니다.

AWS Payment Cryptography 키는 둘 이상의 별칭과 동시에 연결할 수 있지만 각 별칭은 단일 키에만 매핑할 수 있습니다.

예를 들어 이 `list-aliases` 출력은 `alias/sampleAlias1` 별칭이 `KeyArn` 속성으로 표시되는 정확히 하나의 대상 AWS Payment Cryptography 키와 연결되어 있음을 보여줍니다.

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

```
{
  "Aliases": [
      {
          "AliasName": "alias/sampleAlias1",
```

```
            "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
        }
    ]
}
```

여러 별칭을 동일한 AWS Payment Cryptography 키와 연결할 수 있습니다.

예를 들어, `alias/sampleAlias1;` 및 `alias/sampleAlias2` 별칭을 동일한 키로 연결할 수 있습니다.

```
$ 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"
            }
        ]
    }
```

별칭은 해당 계정 및 리전에 대해 고유해야 합니다.

예를 들어, 각 계정 및 리전에서 오직 한 개의 `alias/sampleAlias1` 별칭만 가질 수 있습니다. 별칭은 대소문자를 구분하지만 오류가 발생하기 쉬우므로 대소문자만 다른 별칭은 사용하지 않는 것이 좋습니다. 별칭 이름은 변경할 수 없습니다. 그러나 별칭을 삭제하고 원하는 이름으로 새 별칭을 생성할 수 있습니다.

다른 리전에서 같은 이름으로 별칭을 만들 수 있습니다.

예를 들어, 미국 동부(버지니아 북부)에 `alias/sampleAlias2` 별칭이 있고 미국 서부(오레곤)에 `alias/sampleAlias2` 별칭이 있을 수 있습니다. 각 별칭은 해당 리전의 AWS Payment Cryptography 키와 연결됩니다. 코드가 `alias/finance-key`와 같은 별칭 이름을 참조하는 경우

여러 리전에서 실행할 수 있습니다. 각 리전에서는 다른 별칭/sampleAlias2을 사용합니다. 자세한 내용은 애플리케이션에서 별칭 사용을 참조하세요.

별칭과 연결된 AWS Payment Cryptography 키를 변경할 수 있습니다.

UpdateAlias 작업을 사용하여 별칭을 다른 AWS Payment Cryptography 키와 연결할 수 있습니다. 예를 들어 alias/sampleAlias2별칭이 arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h AWS Payment Cryptography 키와 연결된 경우 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi 키와 연결되도록 업데이트할 수 있습니다.

> **Warning**
>
> AWS Payment Cryptography는 이전 키와 새 키의 키 사용과 같은 속성이 모두 동일한지 검증하지 않습니다. 다른 키 유형으로 업데이트하면 애플리케이션에 문제가 발생할 수 있습니다.

일부 키에는 별칭이 없습니다.

별칭은 선택적 기능이며 이러한 방식으로 환경을 운영하도록 선택하지 않는 한 모든 키에 별칭이 있는 것은 아닙니다. create-alias 명령을 사용하여 키를 별칭과 연결할 수 있습니다. 또한 update-alias 작업을 사용하여 별칭과 연결된 AWS Payment Cryptography 키를 변경하고 update-alias 작업을 사용하여 별칭을 삭제할 수 있습니다. 따라서 일부 AWS Payment Cryptography 키에는 여러 개의 별칭이 있을 수 있으며 일부는 없을 수 있습니다.

키를 별칭에 매핑하기

create-alias 명령을 사용하여 키(ARN으로 표시됨)를 하나 이상의 별칭에 매핑할 수 있습니다. 이 명령은 멱등성이 없습니다. 별칭을 업데이트하려면 update-alias 명령을 사용하세요.

```
$ 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"
```

```
    }
}
```

## 애플리케이션에서 별칭 사용

별칭을 사용하여 애플리케이션 코드에서 AWS Payment Cryptography 키를 나타낼 수 있습니다. AWS Payment Cryptography 데이터 작업 및 List Keys와 같은 다른 작업의 `key-identifier` 파라미터는 별칭 이름 또는 별칭 ARN을 수락합니다.

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

별칭 ARN을 사용하는 경우 AWS Payment Cryptography 키에 대한 별칭 매핑은 AWS Payment Cryptography 키를 소유한 계정에 정의되며 리전마다 다를 수 있습니다.

별칭의 가장 강력한 용도 중 하나는 여러 AWS 리전에서 실행되는 애플리케이션에서 사용하는 것입니다.

각 리전에서 다른 버전의 애플리케이션을 생성하거나 사전, 구성 또는 전환 문을 사용하여 각 리전에 적합한 AWS Payment Cryptography 키를 선택할 수 있습니다. 그러나 각 리전에 동일한 별칭 이름으로 별칭을 만드는 것이 더 쉬울 수 있습니다. 별칭 이름은 대/소문자를 구분합니다.

## 관련 API

Tags

태그는 AWS Payment Cryptography 키를 구성하기 위한 메타데이터 역할을 하는 키 및 값 페어입니다. 키를 유연하게 식별하거나 하나 이상의 키를 그룹화하는 데 사용할 수 있습니다.

# 키 가져오기

AWS Payment Cryptography 키는 암호화 구성 요소의 단일 단위를 나타내며이 서비스의 암호화 작업에만 사용할 수 있습니다. GetKeys API는 KeyIdentifier를 입력으로 받아 속성, 상태 및 타임스탬프를 포함한 키 메타데이터를 반환하지만 실제 암호화 키 구성 요소는 반환하지 않습니다.

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"
  }
}
```

# 키 페어와 연결된 퍼블릭 키/인증서 가져오기

퍼블릭 키/인증서 가져오기는 KeyArn로 표시된 퍼블릭 키를 반환합니다. Payment AWS Cryptography에서 생성된 키 페어의 퍼블릭 키 부분 또는 이전에 가져온 퍼블릭 키일 수 있습니다. 가장 일반적인 사용 사례는 데이터를 암호화하는 외부 서비스에 퍼블릭 키를 제공하는 것입니다. 그런 다음 Payment Cryptography를 활용하여 해당 데이터를 애플리케이션에 전달하고 AWS Payment AWS Cryptography 내에서 보호되는 프라이빗 키를 사용하여 데이터를 복호화할 수 있습니다.

이 서비스는 퍼블릭 키를 공개 인증서로 반환합니다. API 결과에는 CA와 퍼블릭 키 인증서가 포함됩니다. 두 데이터 요소 모두 base64로 인코딩됩니다.

> **Note**
>
> 반환된 공개 인증서는 단기적인 것이며 멱등성을 갖지 않습니다. 퍼블릭 키 자체가 변경되지 않았더라도 각 API 직접 호출마다 다른 인증서를 받을 수 있습니다.

Example

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

```
{
  "KeyCertificate":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2VENDQXFXZ0F3SUJBZ0lSQUo1OWd2VkpDd3dlYldMNldYZEpYY
  "KeyCertificateChain":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUY0VENDQThtZ0F3SUJBZ0lSQUtlN2piaHFKZjJPd3FGUWI5c3VuC
}
```

# 키 태그 지정

AWS Payment Cryptography에서는 키를 생성할 때 AWS Payment Cryptography 키에 태그를 추가하고 삭제 보류 중인 경우가 아니면 기존 키에 태그를 지정하거나 태그를 해제할 수 있습니다. 태그는 선택 사항이지만 매우 유용할 수 있습니다.

모범 사례, 태그 지정 전략, 태그의 형식 및 구문을 포함하여 태그에 대한 일반적인 정보는의 AWS 리소스 태그 지정을 참조하세요Amazon Web Services 일반 참조.

주제

- AWS Payment Cryptography의 태그 정보
- 콘솔에서 키 태그 보기
- API 작업으로 키 태그 관리
- 태그에 대한 액세스 제어
- 태그를 사용하여 키에 대한 액세스 제어

## AWS Payment Cryptography의 태그 정보

태그는 AWS 리소스에 할당(또는 AWS 할당)할 수 있는 선택적 메타데이터 레이블입니다. 각 태그는태그 키 및 태그 값으로 구성되며, 둘 다 대소 문자를 구분하는 문자열입니다. 태그 값은 빈(null) 문자열일 수도 있습니다. 리소스의 각 태그에는 서로 다른 태그 키가 있어야 하지만 여러 AWS 리소스에 동일한 태그를 추가할 수 있습니다. 각 리소스에는 최대 50개의 사용자 생성 태그가 포함될 수 있습니다.

태그 키 또는 태그 값에 기밀 또는 민감한 정보를 포함하지 마세요. 태그는 결제를 AWS 서비스포함하여 많은 사용자가 액세스할 수 있습니다.

AWS Payment Cryptography에서는 키를 생성할 때 키에 태그를 추가하고 삭제 보류 중인 키가 아니면 기존 키에 태그를 지정하거나 태그를 해제할 수 있습니다. 별칭에는 태그를 지정할 수 없습니다. 태그는 선택 사항이지만 매우 유용할 수 있습니다.

예를 들어 Alpha 프로젝트에 사용하는 모든 AWS Payment Cryptography 키와 Amazon S3 버킷에 `"Project"="Alpha"` 태그를 추가할 수 있습니다. 또 다른 예는 특정 은행 식별 번호 (BIN) 와 관련된 모든 키에 `"BIN"="20130622"` 태그를 추가하는 것입니다.

```
    [
```

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

형식 및 구문을 포함한 태그에 대한 일반적인 내용은의 AWS 리소스 태그 지정을 참조하세요Amazon Web Services 일반 참조.

태그는 다음을 지원합니다.

- AWS 리소스를 식별하고 구성합니다. 많은 AWS 서비스가 태그 지정을 지원하므로 서로 다른 서비스의 리소스에 동일한 태그를 할당하여 리소스가 관련이 있음을 나타낼 수 있습니다. 예를 들어 AWS Payment Cryptography 키와 Amazon Elastic Block Store(Amazon EBS) 볼륨 또는 AWS Secrets Manager 보안 암호에 동일한 태그를 할당할 수 있습니다. 태그를 사용하여 자동화를 위해 키를 식별할 수도 있습니다.
- AWS 비용을 추적합니다. AWS 리소스에 태그를 추가하면는 태그별로 집계된 사용량 및 비용이 포함된 비용 할당 보고서를 AWS 생성합니다. 이 기능을 사용하여 프로젝트, 애플리케이션 또는 비용 센터의 AWS Payment Cryptography 비용을 추적할 수 있습니다.

  비용 할당 태그 사용에 대한 자세한 내용은 AWS Billing 사용자 설명서의 비용 할당 태그 사용을 참조하세요. 태그 키 및 태그 값에 대한 규칙에 대한 자세한 내용은 AWS Billing 사용 설명서의 사용자 정의 태그 제한을 참조하세요.
- AWS 리소스에 대한 액세스를 제어합니다. 태그를 기반으로 키에 대한 액세스를 허용하고 거부하는 것은 속성 기반 액세스 제어(ABAC)에 대한 AWS Payment Cryptography 지원의 일부입니다. 태그에 기반으로 AWS Payment Cryptography에 대한 액세스를 제어하는 방법에 대한 자세한 내용은 AWS Payment Cryptography 태그 기반 권한 부여을 참조하세요. 태그를 사용하여 AWS 리소스에 대한 액세스를 제어하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS 리소스 태그를 사용하여 리소스에 대한 액세스 제어를 참조하세요.

AWS Payment Cryptography는 TagResource, UntagResource 또는 ListTagsForResource 작업을 사용할 때 AWS CloudTrail 로그에 항목을 기록합니다.

## 콘솔에서 키 태그 보기

콘솔에서 태그를 보려면 키가 포함된 IAM 정책의 키에 대한 태그 지정 권한이 필요합니다. 콘솔에서 키를 볼 수 있는 권한과 함께 이러한 권한이 필요합니다.

## API 작업으로 키 태그 관리

AWS Payment Cryptography API를 이용해 현재 관리 중인 키에 대한 태그를 추가, 삭제, 나열할 수 있습니다. 이 예제들은 AWS Command Line Interface (AWS CLI)를 사용하지만, 사용자는 어떤 지원되는 프로그래밍 언어라도 사용할 수 있습니다. 태그를 지정할 수 없습니다 AWS 관리형 키.

키의 태그를 추가, 편집, 보기 및 삭제하려면 필요한 사용 권한이 있어야 합니다. 자세한 내용은 태그에 대한 액세스 제어을 참조하세요.

주제

- CreateKey: 새 키에 태그 추가
- TagResource: 키에 태그 추가 또는 변경
- ListResourceTags: 키에 지정된 태그 가져오기
- UntagResource: 키에서 태그 삭제

### CreateKey: 새 키에 태그 추가

키를 생성할 때 태그를 추가할 수 있습니다. 태그를 지정하려면 CreateKey 작업의 Tags 파라미터를 사용합니다.

키를 생성할 때 태그를 추가하려면 IAM 정책에서 호출자가 payment-cryptography:TagResource 권한을 가지고 있어야 합니다. 최소한 권한에 계정 및 리전의 모든 키가 포함되어야 합니다. 자세한 내용은 태그에 대한 액세스 제어을 참조하세요.

CreateKey 의 값은Tags 파라미터 값은 대소문자를 구분하는 태그 키 및 태그 값 페어 모음입니다. 키의 각 태그에는 다른 태그 이름이 있어야 합니다. 태그 값은 null이거나 빈 문자열일 수 있습니다.

예를 들어 다음 AWS CLI 명령은 Project:Alpha 태그가 있는 대칭 암호화 키를 생성합니다. 두 개 이상의 키-값 페어를 지정할 때는 공백을 사용하여 각 페어를 구분합니다.

```
$ 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"}]'
```

이 명령이 성공하면 새 키에 대한 정보가 있는 Key 객체를 반환합니다. 그러나 Key에는 태그가 포함되지 않습니다. 태그를 가져오려면 ListResourceTags 작업을 사용합니다.

## TagResource: 키에 태그 추가 또는 변경

TagResource 작업은 키에 하나 이상의 태그를 추가합니다. 이 작업을 사용하여 다른 AWS 계정의 태그를 추가 또는 편집할 수 없습니다.

태그를 추가하려면 새 태그 키와 태그 값을 지정합니다. 태그를 편집하려면 기존 태그 키와 새 태그 값을 지정합니다. 키의 각 태그에는 다른 태그 키가 있어야 합니다. 태그 값은 null이거나 빈 문자열일 수 있습니다.

예를 들어 다음 명령은 예제 키에 **UseCase** 및 **BIN** 태그를 추가합니다.

```
$ 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"}]'
```

이 명령이 제대로 실행되면 메타데이터를 반환하지 않습니다. 키에 지정된 태그를 보려면 ListResourceTags 작업을 사용합니다.

또한 TagResource를 이용해 기존 태그의 태그 값을 변경할 수도 있습니다. 태그 값을 바꾸려면 동일한 태그 키를 다른 값으로 지정하세요. 수정 명령에 나열되지 않은 태그는 변경되거나 제거되지 않습니다.

예를 들어 이 명령은 Project 태그 값을 Alpha에서 Noe로 바꿉니다.

이 명령은 내용 없이 http/200을 반환합니다. 변경 내용을 보려면 ListTagsForResource을 사용하세요.

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

## ListResourceTags: 키에 지정된 태그 가져오기

ListResourceTags 작업은 키에 지정된 태그를 가져옵니다. ResourceArn(keyArn 또는 keyAlias) 파라미터가 필요합니다. 이 작업을 사용하여 다른 AWS 계정의 키에 있는 태그를 볼 수 없습니다.

예를 들어 다음 명령은 예제 키에 대한 태그를 가져옵니다.

```
$ 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: 키에서 태그 삭제

UntagResource 작업은 키에서 태그를 삭제합니다. 삭제할 태그를 식별하려면 태그 키를 지정합니다. 이 작업을 사용하여 다른 AWS 계정의 키에서 태그를 삭제할 수 없습니다.

성공하면 UntagResource 작업은 어떠한 출력도 반환하지 않습니다. 또한 지정된 태그 키가 키에서 발견되지 않으면 예외를 발생시키거나 응답을 반환하지 않습니다. 작업이 작동했는지 확인하려면 ListResourceTags 작업을 수행합니다.

예를 들어 이 명령은 키에서 **Purpose** 태그와 해당 값을 삭제합니다.

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

## 태그에 대한 액세스 제어

API를 사용하여 태그를 추가하고, 보고, 삭제하려면 보안 주체에게 IAM 정책의 태그 지정 권한이 필요합니다.

태그에 AWS 전역 조건 키를 사용하여 이러한 권한을 제한할 수도 있습니다. AWS Payment Cryptography에서 이러한 조건은 태그 지정 작업(예: TagResource 및 UntagResource)에 대한 액세스를 제어할 수 있습니다.

예제 정책과 자세한 내용은 IAM 사용 설명서의 태그 키를 기반으로 액세스 제어 섹션을 참조하십시오.

태그를 만들고 관리할 수 있는 권한은 다음과 같습니다.

payment-cryptography:TagResource

보안 주체가 태그를 추가하거나 편집할 수 있습니다. 키를 생성하는 동안 태그를 추가하려면 보안 주체에 특정 키로 제한되지 않는 IAM 정책에 대한 권한이 있어야 합니다.

payment-cryptography:ListTagsForResource

보안 주체가 키의 태그를 볼 수 있도록 허용합니다.

payment-cryptography:UntagResource

보안 주체가 키에서 태그를 삭제할 수 있도록 허용합니다.

## 정책에서 태그 지정 권한

키 정책 또는 IAM 정책에서 태그 지정 권한을 제공할 수 있습니다. 예를 들어 다음 예제 키 정책은 키에 대한 태그 지정 권한을 사용자에게 제공합니다. 예를 들어 관리자 또는 개발자 역할로 가정할 수 있는 모든 사용자에게 태그를 볼 수 있는 권한을 제공합니다.

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": "*"
    }
  ]
}
```

보안 주체에게 여러 키에 대한 태그 지정 권한을 부여하려면 IAM 정책을 사용할 수 있습니다. 이 정책이 유효하려면 각 키의 키 정책으로 인해 계정이 IAM 정책을 사용하여 키에 대한 액세스를 제어할 수 있어야 합니다.

예를 들어, 다음 IAM 정책은 보안 주체가 키를 생성할 수 있도록 허용합니다. 또한 지정된 계정의 모든 키에 태그를 만들고 관리할 수 있습니다. 이 조합을 통해 보안 주체가 CreateKey 작업의 태그 파라미터를 사용하여 KMS 키를 만드는 동안 키에 태그를 추가할 수 있습니다.

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/*"
    }
  ]
}
```

## 태그 지정 권한 제한

정책 조건을 사용하여 태그 지정 권한을 제한할 수 있습니다. 다음 정책 조건을 payment-cryptography:TagResource 및 payment-cryptography:UntagResource 권한에 적용할 수 있습니다. 예를 들어, aws:RequestTag/*tag-key* 조건을 사용하여 보안 주체가 특정 태그만 추가하거나 보안 주체가 특정 태그 키를 사용하여 태그를 추가하지 못하도록 할 수 있습니다.

- aws:RequestTag
- aws:ResourceTag/tag-key(IAM 정책만 해당)
- aws:TagKeys

태그를 사용하여 키에 대한 액세스를 제어할 때 가장 좋은 방법은 aws:RequestTag/*tag-key* 또는 aws:TagKeys 조건 키를 사용하여 허용되는 태그 (또는 태그 키)를 결정하는 것입니다.

예를 들어 다음 IAM 정책은 이전 것과 비슷합니다. 그러나 이 정책은 보안 주체가 Project 태그 키가 있는 태그에 대해서만 태그(TagResource)를 생성하고 태그 UntagResource를 삭제할 수 있도록 허용합니다.

왜냐하면 TagResource 및 UntagResource 요청에는 여러 태그가 포함될 수 있으므로 ForAllValues 또는 ForAnyValue 집합 연산자와 aws:TagKeys 조건을 지정해야 합니다. ForAnyValue 연산자를 사용하려면 요청의 태그 키 중 적어도 하나가 정책의 태그 키 중 하나와 일치해야 합니다. ForAllValues 연산자를 사용하려면 요청의 모든 태그 키가 정책의 태그 키 중 하나와 일치해야 합니다. ForAllValues 연산자는 요청에 태그가 없는 경우에도 true를 반환하지만 태그가 지정되지 않으면 TagResource 및 UntagResource가 실패합니다. 집합 연산자에 대한 자세한 내용은 IAM 사용 설명서의 여러 키 및 값 사용을 참조하십시오.

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"}
      }
    }
  ]
}
```

## 태그를 사용하여 키에 대한 액세스 제어

키의 태그를 기반으로 AWS Payment Cryptography에 대한 액세스를 제어할 수 있습니다. 예를 들어 보안 주체가 특정 태그가 있는 키만 활성화 및 비활성화할 수 있도록 허용하는 IAM 정책을 작성할 수 있습니다. 또는 IAM 정책을 사용하여 키에 특정 태그가 없으면 보안 주체가 암호화 작업에서 키를 사용하지 못하도록 할 수 있습니다.

이 기능은 속성 기반 액세스 제어(ABAC)에 대한 AWS Payment Cryptography 지원의 일부입니다. 태그를 사용하여 리소스에 대한 액세스를 제어하는 방법에 대한 자세한 내용은 ABAC란 무엇입니까?를

AWS 참조하세요. AWS 및 IAM 사용 설명서의 AWS 리소스 태그를 사용하여 리소스에 대한 액세스 제어.

AWS Payment Cryptography는 키의 태그를 기반으로 키에 대한 액세스를 제어할 수 있는 aws:ResourceTag/tag-key 전역 조건 컨텍스트 키를 지원합니다. 여러 키가 동일한 태그를 가질 수 있으므로 이 기능을 사용하면 선택한 키 집합에 사용 권한을 적용할 수 있습니다. 태그를 변경하여 집합의 키를 쉽게 변경할 수도 있습니다.

AWS Payment Cryptography에서 `aws:ResourceTag/`*`tag-key`* 조건 키는 IAM 정책에서만 지원됩니다. 하나의 키에만 적용되는 키 정책이나 ListKeys 또는 ListAliases 작업처럼 특정 키를 사용하지 않은 작업에서는 지원되지 않습니다.

태그를 사용하여 액세스를 제어하면 사용 권한을 단순하고 확장 가능하며 유연하게 관리할 수 있습니다. 그러나 제대로 설계되고 관리되지 않으면 실수로 키에 대한 액세스를 허용하거나 거부할 수 있습니다. 태그를 사용하여 액세스를 제어하는 경우 다음 방법을 고려하세요.

- 최소 권한 액세스를 강화하는 최고의 방식은 태그를 사용하는 것입니다. IAM 보안 주체에 사용하거나 관리해야 하는 키에만 필요한 권한만 부여합니다. 예를 들어 태그를 사용하여 프로젝트에 사용되는 키에 레이블을 지정합니다. 그런 다음 프로젝트 팀에 프로젝트 태그와 함께 키만 사용할 수 있는 권한을 부여합니다.
- 보안 주체에게 태그를 추가, 편집 및 삭제할 수 있는 `payment-cryptography:TagResource` 및 `payment-cryptography:UntagResource` 권한을 부여할 때는 주의해야 합니다. 태그를 사용하여 키에 대한 액세스를 제어하는 경우 태그를 변경하면 보안 주체에게 사용 권한이 없는 키를 사용할 수 있는 권한이 부여될 수도 있습니다. 또한 다른 보안 주체가 작업을 수행하는 데 필요한 키에 대한 액세스를 거부할 수도 있습니다. 키 정책을 변경하거나 권한 부여를 생성할 권한이 없는 키 관리자는 태그를 관리할 권한이 있는 경우 키에 대한 액세스를 제어할 수 있습니다.

  가능하면 정책 조건(예: `aws:RequestTag/`*`tag-key`* 또는 `aws:TagKeys`)을 사용하여 보안 주체의 태그 지정 권한을를 특정 키의 특정 태그 또는 태그 패턴으로 제한합니다.
- 현재 태그 지정 및 태그 해제 권한이 AWS 계정 있는의 보안 주체를 검토하고 필요한 경우 해당 보안 주체를 조정합니다. IAM 정책은 모든 키에 대한 태그 및 태그 해제 권한을 허용할 수 있습니다. 예를 들어 관리자 관리형 정책은 보안 주체가 모든 키에 대해 태그를 지정하고 태그를 해제하고 나열할 수 있도록 허용합니다.
- 태그에 의존하는 정책을 설정하기 전에의 키에서 태그를 검토합니다 AWS 계정. 포함하려는 태그에만 정책을 적용해야 합니다. CloudTrail 로그 및 CloudWatch 경보를 사용하여 키에 대한 액세스에 영향을 줄 수 있는 태그 변경 사항을 알립니다.
- 태그 기반 정책 조건은 패턴 일치를 사용하며 태그의 특정 인스턴스에 연결되어 있지 않습니다. 태그 기반 조건 키를 사용하는 정책은 패턴과 일치하는 모든 새 태그와 기존 태그에 영향을 줍니다. 정책

조건과 일치하는 태그를 삭제했다가 다시 만들면 이전 태그와 마찬가지로 조건이 새 태그에 적용됩니다.

예를 들어 다음과 같은 IAM 정책을 살펴보십시오. 이를 통해 보안 주체는 사용자 계정의 키 중 미국 동부 (버지니아 북부) 리전이고 "Project"="Alpha" 태그가 있는 키에 대해서만 암호 해독 작업을 호출할 수 있습니다. 이 정책은 예제 Alpha 프로젝트의 역할에 연결할 수 있습니다.

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"
        }
      }
    }
  ]
}
```

다음 예제 IAM 정책은 보안 주체가 특정 암호화 작업에 대해 계정의 모든 키를 사용하도록 허용합니다. 그러나 보안 주체가 "Type"="Reserved" 태그가 있거나 "Type" 태그가 없는 키에서 이러한 암호화 작업을 사용하는 것은 허용되지 않습니다.

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"
        }
      }
    }
  ]
}
```

# AWS Payment Cryptography 키의 키 속성 이해

적절한 키 관리의 원칙은 키의 범위가 적절하게 지정되고 허용된 작업에만 사용할 수 있어야 한다는 것입니다. 따라서 특정 키는 특정 키 사용 모드에서만 생성할 수 있습니다. 이는 가능한 경우 TR-31 정의에 따라 사용 가능한 사용 모드에 맞게 조정됩니다.

AWS Payment Cryptography에서는 잘못된 키를 생성할 수 없지만 사용자의 편의를 위해 유효한 조합이 여기에 제공됩니다.

## 대칭 키

- TR31_B0_BASE_DERIVATION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 키 사용 모드 조합: { DeriveKey = true },{ NoRestrictions = true }
- TR31_C0_CARD_VERIFICATION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128* ,AES_192* ,AES_256*
  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } ,{ NoRestrictions = true }
- TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY*, AES_128* ,AES_192* ,AES_256*
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }, { NoRestrictions = true }
- TR31_E1_EMV_MKEY_CONFIDENTIALITY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY, AES_128*,AES_192*,AES_256*
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }, { NoRestrictions = true }
- TR31_E2_EMV_MKEY_INTEGRITY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128* ,AES_192* ,AES_256*
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }, { NoRestrictions = true }
- TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128* ,AES_192* ,AES_256*

  - 허용되는 키 사용 모드 조합: { DeriveKey = true }, { NoRestrictions = true }
- TR31_E5_EMV_MKEY_CARD_PERSONALIZATION
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128* ,AES_192* ,AES_256*
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }, { NoRestrictions = true }
- TR31_E6_EMV_MKEY_OTHER
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128* ,AES_192* ,AES_256*
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }, { NoRestrictions = true }
- TR31_K0_KEY_ENCRYPTION_KEY
  - TR31_K1_KEY_BLOCK_PROTECTION_KEY를 사용하는 것이 좋습니다. 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } ,{ NoRestrictions = true }
- TR31_K1_KEY_BLOCK_PROTECTION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } ,{ NoRestrictions = true }
- TR31_M1_ISO_9797_1_MAC_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY
  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_M3_ISO_9797_3_MAC_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY
  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_M6_ISO_9797_5_CMAC_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_M7_HMAC_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256

  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_P0_PIN_ENCRYPTION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } ,{ NoRestrictions = true }
- TR31_V1_IBM3624_PIN_VERIFICATION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_V2_VISA_PIN_VERIFICATION_KEY
  - 허용된 키 알고리즘: TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 허용되는 주요 사용 모드 조합: { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }

## 비대칭 키

- TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION
  - 허용된 키 알고리즘: RSA_2048 ,RSA_3072 ,RSA_4096
  - 허용되는 주요 사용 모드 조합: { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true }
  - 참고: { Encrypt = true, Wrap = true }는 데이터를 암호화하거나 키를 래핑하기 위한 퍼블릭 키를 가져올 때 유일하게 유효한 옵션입니다.
- TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE
  - 허용된 키 알고리즘: RSA_2048 ,RSA_3072 ,RSA_4096
  - 허용된 키 사용 모드 조합: { Sign = true } ,{ Verify = true }
  - 참고: { Verify = true }는 루트 인증서, 중간 인증서 또는 TR-34용 서명 인증서와 같이 서명용 키를 가져올 때 유일하게 유효한 옵션입니다.
- TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT
  - ECDH와 같은 주요 계약 알고리즘에 사용됩니다.
  - 허용된 키 알고리즘: ECC_NIST_P256,ECC_NIST_P384,ECC_NIST_P521
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }.

  - 참고:DeriveKeyUsage는이 기본 키에서 파생될 키 종류를 지정하는 데 사용됩니다. 키 생성/가져오기 시 수정됩니다.
- TR31_K2_TR34_ASYMMETRIC_KEY
  - TR-34와 같은 X9.24 호환 키 교환 메커니즘에 사용되는 비대칭 키
  - 허용된 키 알고리즘: RSA_2048,RSA_3072,RSA_4096
  - 허용되는 키 사용 모드 조합: { DeriveKey = true }.
  - 허용되는 주요 사용 모드 조합: { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true }
  - 참고: { Encrypt = true, Wrap = true }는 데이터를 암호화하거나 키를 래핑하기 위한 퍼블릭 키를 가져올 때 유일하게 유효한 옵션입니다.

*이 알고리즘/키 유형 조합은 현재 암호화 작업에서 지원되지 않습니다.

# 데이터 작업

AWS Payment Cryptography 키를 설정한 후 암호화 작업을 수행하는 데 사용할 수 있습니다. 다양한 작업을 통해 암호화, 해싱, CVV2 생성과 같은 도메인별 알고리즘에 이르기까지 다양한 유형의 활동을 수행합니다.

암호화된 데이터는 매칭된 암호 해독 키(암호화 유형에 따라 대칭 키 또는 프라이빗 키)가 없으면 해독할 수 없습니다. 대칭 키 또는 퍼블릭 키 없이는 해싱 및 도메인별 알고리즘을 유사하게 확인할 수 없습니다.

특정 작업에 유효한 키 유형에 대한 자세한 내용은 암호화 작업에 유효한 키를 참조하세요.

> **Note**
>
> 비프로덕션 환경에서는 테스트 데이터를 사용하는 것이 좋습니다. 비프로덕션 환경에서 프로덕션 키와 데이터(PAN, BDK ID 등)를 사용하면 PCI DSS 및 PCI P2PE와 같은 규정 준수 범위에 영향을 미칠 수 있습니다.

주제


- 데이터 암호화, 복호화 및 재암호화
- 카드 데이터 생성 및 확인
- PIN 데이터 생성, 변환 및 확인
- 인증 요청(ARQC) 암호화 확인
- HMAC 생성 및 확인
- 암호화 작업을 위한 유효한 키


## 데이터 암호화, 복호화 및 재암호화

암호화 및 해독 방법은 TDES, AES 및 RSA를 비롯한 다양한 대칭 및 비대칭 기술을 사용하여 데이터를 암호화하거나 해독하는 데 사용될 수 있습니다. 이러한 메서드는 DUKPT 및 EMV 기술을 사용하여 파생된 키도 지원합니다. 기본 데이터를 노출하지 않고 새 키로 데이터를 보호하려는 사용 사례의 경우 ReEncrypt 명령을 사용할 수도 있습니다.

> **Note**
>
> 암호화/복호화 함수를 사용하는 경우 모든 입력은 hexBinary로 간주됩니다. 예를 들어 값 1은 31(16진수)로 입력되고 소문자 t는 74(16진수)로 표시됩니다. 모든 출력값도 hexBinary로 표시됩니다.

사용 가능한 모든 옵션에 대한 자세한 내용은 API Guide for Encrypt, Decrypt, and Re-Encrypt를 참조하세요.

주제

- Encrypt data
- 데이터 해독

## Encrypt data

`Encrypt Data` API는 대칭 및 비대칭 데이터 암호화 키와 DUKPT 및 EMV 파생 키를 사용하여 데이터를 암호화하는 데 사용됩니다. TDES, RSA, AES를 비롯한 다양한 알고리즘과 변형이 지원됩니다.

기본 입력은 데이터를 암호화하는 데 사용되는 암호화 키, 암호화할 hexBinary 형식의 일반 텍스트 데이터, TDES와 같은 블록 암호의 초기화 벡터 및 모드와 같은 암호화 속성입니다. 일반 텍스트 데이터는의 경우 8바이트TDES,의 경우 16바이트AES,의 경우 키 길이의 배수여야 합니다RSA. 입력 데이터가 이러한 요구 사항을 충족하지 않는 경우 대칭 키 입력(TDES, AES, DUKPT, EMV)을 패딩해야 합니다. 다음 표에는 각 키 유형에 대한 일반 텍스트의 최대 길이와 RSA 키에 `EncryptionAttributes` 대해에서 정의하는 패딩 유형이 나와 있습니다.

| 패딩 유형 | 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 |

기본 출력값에는 hexBinary 형식의 사이퍼텍스트로 암호화된 데이터와 암호화 키의 체크섬 값이 포함됩니다. 사용 가능한 모든 옵션에 대한 자세한 내용은 암호화를 위한 API 가이드를 참조하세요.

예제

- AES 대칭 키를 사용하여 데이터를 암호화합니다.
- DUKPT 키를 사용한 데이터 암호화
- EMV 파생 대칭 키를 사용하여 데이터 암호화
- RSA 키를 사용한 데이터 암호화

## AES 대칭 키를 사용하여 데이터를 암호화합니다.

> **Note**
>
> 모든 예제에서는 관련 키가 이미 있다고 가정합니다. 키는 CreateKey 작업을 사용하여 생성하거나 ImportKey 작업을 사용하여 가져올 수 있습니다.

Example

이 예시에서는 CreateKey 작업을 사용하여 생성했거나 ImportKey 작업을 사용하여 가져온 대칭 키를 사용하여 일반 텍스트 데이터를 암호화합니다. 이 작업을 수행하려면 키의 KeyModesOfuse를 `Encrypt`로 설정하고 KeyUse를 `TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY`로 설정해야 합니다. 자세한 옵션은 암호화 작업을 위한 키를 참조하세요.

```
$ 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"
}
```

## DUKPT 키를 사용한 데이터 암호화

Example

이 예제에서는 DUKPT 키를 사용하여 일반 텍스트 데이터를 암호화합니다. AWS Payment Cryptography는 TDES 및 AES DUKPT 키를 지원합니다. 이 작업을 수행하려면 키의 KeyModesOfuse를 DeriveKey로 설정하고 KeyUse를 TR31_B0_BASE_DERIVATION_KEY로 설정해야 합니다. 자세한 옵션은 암호화 작업을 위한 키를 참조하세요.

```
$ 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"
}
```

## EMV 파생 대칭 키를 사용하여 데이터 암호화

Example

이 예제에서는 이미 생성된 EMV 파생 대칭 키를 사용하여 일반 텍스트 데이터를 암호화합니다. 이와 같은 명령을 사용하여 EMV 카드에 데이터를 전송할 수 있습니다. 이 작업을 수행하려면 키에 KeyModesOfUse가 로 설정되어 Derive 있고 KeyUsage가 TR31_E1_EMV_MKEY_CONFIDENTIALITY 또는 로 설정되어 있어야 합니다TR31_E6_EMV_MKEY_OTHER. 자세한 내용은 암호화 작업용 키를 참조하세요.

```
$ 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"
}
```

## RSA 키를 사용한 데이터 암호화

Example

이 예시에서는 ImportKey 작업을 사용하여 가져온 RSA 퍼블릭 키를 사용하여 일반 텍스트 데이터를 암호화합니다. 이 작업을 수행하려면 키의 KeyModesOfuse를 `Encrypt`로 설정하고 KeyUse를 `TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION`로 설정해야 합니다. 자세한 옵션은 암호화 작업을 위한 키를 참조하세요.

PKCS #7 또는 현재 지원되지 않는 기타 패딩 체계의 경우 서비스를 호출하기 전에 적용하고 패딩 표시기 'Asymmetric={}'을 생략하여 패딩 없음을 선택하세요.

```
$ aws payment-cryptography-data encrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/thfezpmsalcfwmsg
 --plain-text 31323334313233343132333431323334 --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"
}
```

## 데이터 해독

`Decrypt Data` API는 대칭 및 비대칭 데이터 암호화 키와 DUKPT 및 EMV 파생 키를 사용하여 데이터를 복호화하는 데 사용됩니다. TDES, RSA, AES를 비롯한 다양한 알고리즘과 변형이 지원됩니다.

기본 입력값은 데이터를 해독하는 데 사용되는 암호 해독 키, 해독할 hexBinary 형식의 사이퍼텍스트 데이터, 초기화 벡터, 블록 암호 모드 등과 같은 해독 특성입니다. 기본 출력값에는 hexBinary 형식의 일반 텍스트로 해독된 데이터와 암호 해독 키의 체크섬 값이 포함됩니다. 사용 가능한 모든 옵션에 대한 자세한 내용은 Decrypt용 API 가이드를 참조하세요.

예제

- AES 대칭 키를 사용한 데이터 해독
- DUKPT 키를 사용한 데이터 해독
- EMV 파생 대칭 키를 사용하여 데이터 복호화
- RSA 키를 사용한 데이터 해독

## AES 대칭 키를 사용한 데이터 해독

Example

이 예제에서는 대칭 키를 사용하여 사이퍼텍스트 데이터를 복호화합니다. 이 예제에서는 AES 키를 보여주지만 TDES_2KEY 및 도 지원TDES_3KEY됩니다. 이 작업을 수행하려면 키의 KeyModesOfuse를 Decrypt로 설정하고 KeyUse를 TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY로 설정해야 합니다. 자세한 옵션은 암호화 작업을 위한 키를 참조하세요.

```
$ 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"
}
```

# DUKPT 키를 사용한 데이터 해독

> **Note**
>
> P2PE 거래에 DUKPT와 함께 암호 해독 데이터를 사용하면 PCI DSS 범위를 결정할 때 고려해야 하는 신용 카드 PAN 및 기타 카드 소유자 데이터가 애플리케이션에 반환될 수 있습니다.

### Example

이 예시에서는 CreateKey 작업을 사용하여 생성되었거나 ImportKey 작업을 사용하여 가져온 DUKPT 키를 사용하여 사이퍼텍스트 데이터를 해독합니다. 이 작업을 수행하려면 키의 KeyModesOfuse를 DeriveKey로 설정하고 KeyUse를 TR31_B0_BASE_DERIVATION_KEY로 설정해야 합니다. 자세한 옵션은 암호화 작업을 위한 키를 참조하세요. TDES 알고리즘에 DUKPT를 사용하는 경우 사이퍼텍스트 데이터 길이는 16바이트의 배수여야 합니다. AES 알고리즘의 경우 사이퍼텍스트 데이터 길이는 32바이트의 배수여야 합니다.

```
$ 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": "313233343132333431323334313233343"
}
```

## EMV 파생 대칭 키를 사용하여 데이터 복호화

### Example

이 예제에서는 CreateKey 작업을 사용하여 생성되었거나 ImportKey 작업을 사용하여 가져온 EMV 파생 대칭 키를 사용하여 사이퍼텍스트 데이터를 복호화합니다. 이 작업을 수행하려면 키에 KeyModesOfUse가 로 설정되어 `Derive` 있고 KeyUsage가 `TR31_E1_EMV_MKEY_CONFIDENTIALITY` 또는 로 설정되어 있어야 합니다`TR31_E6_EMV_MKEY_OTHER`. 자세한 내용은 암호화 작업용 키를 참조하세요.

```
$ 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"
}
```

### RSA 키를 사용한 데이터 해독

Example

이 예시에서는 CreateKey 작업을 사용하여 생성된 RSA 키 페어를 사용하여 사이퍼텍스트 데이터를 해독합니다. 이 작업을 수행하려면 키의 KeyModesOfUse를 Decrypt 활성화로 설정하고 KeyUse를 TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION로 설정해야 합니다. 자세한 옵션은 암호화 작업을 위한 키를 참조하세요.

PKCS #7 또는 현재 지원되지 않는 다른 패딩 체계의 경우, 패딩 표시기 'Asymmetric={}'을 생략하여 패딩 없음을 선택하고 서비스를 호출한 후 패딩을 제거하세요.

```
$ 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": "313233343132333431323334313233343"
}
```

# 카드 데이터 생성 및 확인

카드 데이터를 생성하고 CVV, CVV2, CVC, DCVV와 같은 카드 데이터에서 파생된 데이터가 카드 데이터에 통합되는지 확인합니다.

주제

- 카드 데이터 생성
- 카드 데이터 확인

# 카드 데이터 생성

`Generate Card Data` API는 CVV, CVV2 또는 동적 CVV2와 같은 알고리즘을 사용하여 카드 데이터를 생성하는 데 사용됩니다. 이 명령에 사용할 수 있는 키를 보려면 암호화 작업에 유효한 키 섹션을 참조하세요.

CVV, CVV2, iCVV, CAVV V7과 같은 많은 암호화 값은 동일한 암호화 알고리즘을 사용하지만 입력 값은 다릅니다. 예를 들어 CardVerificationValue1에는 ServiceCode, 카드 번호 및 만료 날짜 입력이 있습니다. CardVerificationValue2에는 이러한 입력 중 두 개만 있지만 CVV2/CVC2의 경우 ServiceCode가 000으로 고정되기 때문입니다. 마찬가지로 iCVV의 경우 ServiceCode는 999로 고정됩니다. 일부 알고리즘은 CAVV V8과 같은 기존 필드의 용도를 변경할 수 있으며,이 경우 올바른 입력 값은 공급자 설명서를 참조해야 합니다.

> **Note**
>
> 올바른 결과를 생성하려면 생성 및 검증을 위해 만료 날짜를 동일한 형식(예: MMYY 및 YYMM)으로 입력해야 합니다.

## CVV2 생성

Example

이 예제에서는 PAN 및 카드 만료 날짜를 입력하는 지정된 PAN에 대한 CVV2를 생성합니다. 여기서는 카드 확인 키가 생성되었다고 가정합니다.

```
$ 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"
  }
```

## iCVV 생성

Example

이 예제에서는 999의 서비스 코드 및 카드 만료 날짜를 입력하는 지정된 PAN에 대한 iCVVPAN를 생성합니다. 여기서는 카드 확인 키가 생성되었다고 가정합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

```
$ 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"
}
```

## 카드 데이터 확인

`Verify Card Data`는 `DISCOVER_DYNAMIC_CARD_VERIFICATION_CODE`와 같은 암호화 원리에 의존하는 결제 알고리즘을 사용하여 생성된 데이터를 확인하는 데 사용됩니다.

입력 값은 일반적으로 인바운드 트랜잭션의 일부로 발급자 또는 지원 플랫폼 파트너에게 제공됩니다. ARQC 암호화(EMV 칩 카드에 사용)를 확인하려면 ARQC 확인을 참조하세요.

자세한 내용은 API 가이드의 VerifyCardValidationData를 참조하세요.

값이 확인되면 API는 http/200을 반환합니다. 값이 확인되지 않은 경우 http/400을 반환합니다.

## CVV2 확인

### Example

이 예시에서는 주어진 PAN에 대해 CVV/CVV2를 검증해 보겠습니다. CVV2는 일반적으로 카드 소유자 또는 사용자가 거래 시간 동안 검증을 위해 제공합니다. 입력값을 검증하기 위해 런타임 시 검증에 사용할 키(CVK), PAN, 카드 만료일 및 CVV2 입력 등의 값이 제공됩니다. 카드 만료 형식은 초기 값 생성에 사용된 형식과 일치해야 합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue2를 참조하세요.

```
$ 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"
  }
```

### iCVV 확인

Example

이 예제에서는 검증에 사용할 키(CVK), , 999의 서비스 코드PAN, 카드 만료 날짜 및 검증을 위해 트랜잭션에서 제공한 iCVV를 입력하여 지정된 PAN에 대한 iCVV를 확인합니다. ???

iCVV는 사용자가 입력한 값(예: CVV2)이 아니라 EMV 카드에 포함됩니다. 제공 시 항상 검증해야 하는지 여부를 고려해야 합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

```
$ 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"
}
```

## PIN 데이터 생성, 변환 및 확인

PIN 데이터 함수를 사용하면 무작위 핀, 핀 확인 값(PVV)을 생성하고 PVV 또는 PIN 오프셋에 대해 인바운드 암호화 핀을 검증할 수 있습니다.

핀 변환을 사용하면 PCI PIN 요구 사항 1에 지정된 일반 텍스트로 핀을 노출하지 않고도 한 작업 키에서 다른 작업 키로 핀을 변환할 수 있습니다.

> **Note**
>
> PIN 생성 및 검증은 일반적으로 발급자 기능이고 PIN 변환은 일반적인 취득자 기능이므로 권한이 가장 적은 액세스를 고려하고 시스템 사용 사례에 적합한 정책을 설정하는 것이 좋습니다.

주제


- PIN 데이터 변환
- PIN 데이터 생성
- PIN 데이터 확인


## PIN 데이터 변환

PIN 데이터 변환 함수는 암호화된 데이터가 HSM을 벗어나지 않고 한 키 세트에서 다른 키 세트로 암호화된 PIN 데이터를 변환하는 데 사용됩니다. 작업 키는 변경해야 하지만 처리 시스템에서 데이터를 해독할 필요가 없거나 해독이 허용되지 않는 P2PE 암호화에 사용됩니다. 기본 입력값은 암호화된 데이터, 데이터를 암호화하는 데 사용되는 암호화 키, 입력 값을 생성하는 데 사용되는 파라미터입니다. 다른 입력 세트는 출력값을 암호화하는 데 사용되는 키와 해당 출력값을 생성하는 데 사용되는 파라미터와 같은 요청된 출력 파라미터입니다. 기본 출력값은 새로 암호화된 데이터 세트와 이를 생성하는 데 사용된 파라미터입니다.

> **Note**
>
> PCI 규정 준수를 위해 수신 및 발신 PrimaryAccountNumber 값이 일치해야 합니다. 한 PAN에서 다른 PAN으로 PIN을 변환하는 것은 허용되지 않습니다.

주제


- PEK에서 DUKPT로의 PIN
- PEK에서 PEK로의 PIN

## PEK에서 DUKPT로의 PIN

### Example

이 예제에서는 DUKPT를 사용하는 AES ISO 4 PIN 블록의 PIN을 ISO 0 PIN 블록을 사용하는 PEK TDES 암호화로 변환합니다. 이는 결제 터미널이 ISO 4의 핀을 암호화하는 경우 일반적이며, 다음 연결이 아직 AES를 지원하지 않는 경우 다운스트림 처리를 위해 TDES로 다시 변환될 수 있습니다.

```
$ 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"
        }
```

### PEK에서 PEK로의 PIN

Example

이 예제에서는 한 PEK(PIN 암호화 키)로 암호화된 PIN을 다른 PEK로 변환합니다. 이는 일반적으로 서로 다른 암호화 키를 사용하는 여러 시스템 또는 파트너 간에 트랜잭션을 라우팅하는 동시에 프로세스 전반에 걸쳐 PIN을 암호화하여 PCI PIN 규정 준수를 유지하는 데 사용됩니다. 이 예제에서는 두 키 모두 TDES 3KEY 암호화를 사용하지만 AES ISO-4에서 TDES ISO-0으로, DUKPT에서 PEK로, AS2805에서 PEK로 등 다양한 옵션을 사용할 수 있습니다.

```
$ 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"
}
```

이제 출력 PIN 블록이 두 번째 PEK로 암호화되어 해당 키를 보유한 다운스트림 시스템으로 안전하게 전송할 수 있습니다.

## PIN 데이터 생성

PIN 데이터 생성 함수는 트랜잭션 또는 승인 기간 동안 사용자의 핀 입력을 검증하는 데 사용되는 PVV 및 핀 블록 오프셋과 같은 PIN 관련 값을 생성하는 데 사용됩니다. 또한 이 API는 다양한 알고리즘을 사용하여 임의의 새로운 핀을 생성할 수 있습니다.

## 무작위 핀 및 일치하는 Visa PVV 생성

Example

이 예제에서는 출력이 암호화된 (PinData.PinBlock) 및 PIN block (pinData.Offset)가 되는 새 PVV (무작위) 핀을 생성합니다.PinData.PinBlock) pinData.Offset). 키 입력값은PAN, Pin Verification Key, Pin Encryption Key, PIN block format입니다.

이 명령을 사용하려면 키가 유형이어야 합니다TR31_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"
            }
        }
```

## 알려진 핀에 대한 Visa PVV 생성

Example

이 예제에서는 지정된(암호화된) 핀에 대한 PVV를 생성합니다. 암호화된 핀은 사용자가 선택할 수 있는 핀 흐름을 사용하여 결제 터미널 또는 카드 소지자와 같은 업스트림에서 수신될 수 있습니다. 키 입력은 PAN, Pin Verification Key, Pin Encryption Key, Encrypted Pin Block 및 입니다PIN 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"
            }
        }
```

## 핀에 대한 IBM3624 핀 오프셋 생성

IBM 3624 PIN 오프셋을 IBM 메서드라고도 합니다. 이 방법은 검증 데이터(일반적으로 PAN)와 PIN 키(PVK)를 사용하여 자연/중간 PIN을 생성합니다. 자연 핀은 사실상 파생된 값이며 카드 소지자 수준에서 핀 데이터를 저장할 필요가 없으므로 발급자를 처리하는 데 매우 효율적입니다. 가장 확실한 단점은 이 체계가 카드 소지자 선택 가능 핀이나 무작위 핀을 고려하지 않는다는 것입니다. 이러한 유형의 핀을 허용하기 위해 오프셋 알고리즘이 체계에 추가되었습니다. 오프셋은 선택한(또는 무작위) 핀과 자연키 간의 차이를 나타냅니다. 오프셋 값은 카드 발급자 또는 카드 프로세서에 의해 저장됩니다. 트랜잭션 시 AWS Payment Cryptography 서비스는 내부적으로 자연 핀을 다시 계산하고 오프셋을 적용하여 핀을 찾습니다. 그런 다음 이를 트랜잭션 권한 부여에서 제공한 값과 비교합니다.

IBM3624에는 다음과 같은 몇 가지 옵션이 있습니다.

- `Ibm3624NaturalPin`는 자연 핀과 암호화된 핀 블록을 출력합니다.
- `Ibm3624PinFromOffset`는 오프셋을 고려하여 암호화된 핀 블록을 생성합니다.
- `Ibm3624RandomPin`는 무작위 핀을 생성한 다음 일치하는 오프셋과 암호화된 핀 블록을 생성합니다.
- `Ibm3624PinOffset`는 사용자가 선택한 핀을 고려하여 핀 오프셋을 생성합니다.

AWS Payment Cryptography 내부에서 다음 단계를 수행합니다.

- 제공된 팬을 16자로 패딩합니다. <16이 제공된 경우 제공된 패딩 문자를 사용하여 오른쪽에서 패드합니다.
- PIN 생성 키를 사용하여 검증 데이터를 암호화합니다.
- 소수 표를 사용하여 암호화된 데이터를 소수 자릿수화합니다. 그러면 인스턴스 'A'가 9에 매핑되고 1이 1에 매핑될 수 있는 16진수 숫자가 10진수에 매핑됩니다.
- 출력의 16진수 표현에서 처음 4자리를 가져옵니다. 이것이 자연 핀입니다.
- 사용자가 선택하거나 무작위 핀이 생성된 경우 모듈로에서 고객 핀으로 자연 핀을 뺍니다. 결과는 핀 오프셋입니다.

예제

- 예: 핀에 대한 IBM3624 핀 오프셋 생성

예: 핀에 대한 IBM3624 핀 오프셋 생성

이 예제에서는 출력이 암호화된(PinData.PinBlock) 및 `IBM3624` 오프셋 값`PIN block`(pinData.Offset).PinData.PinBlock) 입력은 PAN, 검증 데이터(일반적으로 팬), 패딩 문자, `Pin Verification Key`, `Pin Encryption Key` 및 입니다`PIN block format`.

이 명령을 사용하려면 핀 생성 키가 유형이고 암호화 키가 유형이어야 `TR31_V1_IBM3624_PIN_VERIFICATION_KEY` 합니다. `TR31_P0_PIN_ENCRYPTION_KEY`

Example

다음 예제에서는 Ibm3624RandomPin을 사용하여 무작위 핀을 생성한 다음 암호화된 핀 블록과 IBM3624 오프셋 값을 출력하는 방법을 보여줍니다. 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"
              }
          }
```

## PIN 데이터 확인

PIN 데이터 확인 함수는 PIN이 올바른지 확인하는 데 사용됩니다. 여기에는 일반적으로 이전에 저장된 핀 값을 카드 소유자가 POI에 입력한 값과 비교하는 작업이 포함됩니다. 이러한 함수는 각 소스의 기본 값을 노출하지 않고 두 값을 비교합니다.

## PVV 메서드를 사용하여 암호화된 PIN 검증

Example

이 예시에서는 지정된 PAN의 PIN을 검증해 보겠습니다. PIN은 일반적으로 카드 소지자 또는 사용자가 검증을 위해 제공하며 파일의 값과 비교됩니다(카드 소지자의 입력은 터미널 또는 기타 업스트림 공급자의 암호화된 값으로 제공됨). 이 입력을 검증하기 위해 런타임 시 입력 핀을 암호화하는 데 사용되는 키(종종 )PAN와 확인할 값( PVV 또는 IWK)도 제공됩니다PIN offset.

AWS Payment Cryptography가 핀을 검증할 수 있는 경우 http/200이 반환됩니다. 핀이 검증되지 않은 경우 http/400을 반환합니다.

```
$ 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",

        }
```

## PVV 메서드를 사용하여 암호화된 PIN 검증 - 오류 잘못된 핀

Example

이 예제에서는 지정된 PAN에 대한 PIN 검증을 시도하지만 핀이 잘못되어 실패합니다.

SDKs 사용할 때 {"Message":"Pin 블록 확인 실패.","Reason":"INVALID_PIN"}으로 표시됩니다.

```
$ 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.
```

## PVV 메서드를 사용하여 암호화된 PIN 검증 - 잘못된 입력 오류

Example

이 예제에서는 지정된 PAN에 대한 PIN 검증을 시도하지만 잘못된 입력으로 인해 실패하고 수신 데이터가 유효한 핀이 아닙니다. 일반적인 원인은 다음과 같습니다. 사용 중인 1/잘못된 키 2/팬 또는 핀 블록 형식과 같은 입력 파라미터가 잘못된 3/핀 블록이 손상되었습니다.

```
$ 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.
```

## 이전에 저장된 IBM3624 핀 오프셋과 비교하여 PIN 검증

이 예제에서는 카드 발급자/프로세서와 함께 파일에 저장된 핀 오프셋과 비교하여 카드 소지자가 제공한 PIN을 검증합니다. 입력은 결제 터미널(또는 카드 네트워크???와 같은 다른 업스트림 공급자)에서 제공하는 암호화된 핀의 추가와 유사합니다. 핀이 일치하면 API는 http 200을 반환합니다. 여기서 출력은 암호화된 값PIN block(PinData.PinBlock)과 IBM3624 오프셋 값(pinData.Offset).

이 명령을 사용하려면 핀 생성 키가 유형이고 암호화 키가 유형이어야 TR31_V1_IBM3624_PIN_VERIFICATION_KEY 합니다. TR31_P0_PIN_ENCRYPTION_KEY

Example

```
$ 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"
}
}
```

# 인증 요청(ARQC) 암호화 확인

인증 요청 암호화 API는 ARQC를 확인하는 데 사용됩니다. ARQC 생성은 AWS Payment Cryptography의 범위를 벗어나며 일반적으로 트랜잭션 권한 부여 시간 동안 EMV Chip Card(또는 모바일 지갑과 같은 디지털에 상응하는 카드)에서 수행됩니다. ARQC는 각 트랜잭션마다 고유하며 카드의 유효성을 암호화하여 보여주고 트랜잭션 데이터가 현재(예상) 트랜잭션과 정확히 일치하는지 확인하는 데 사용합니다.

AWS Payment Cryptography는 ARQC를 검증하고 EMV 4.4 Book 2 및 Visa 및 Mastercard에서 사용하는 기타 체계에 정의된 값을 포함하여 선택적 ARPC 값을 생성하기 위한 다양한 옵션을 제공합니다. 사용 가능한 모든 옵션의 전체 목록은 API 안내서의 VerifyCardValidationData 섹션을 참조하세요.

ARQC 암호에는 일반적으로 다음 입력이 필요합니다(구현에 따라 다를 수 있음).

- PAN - PrimaryAccountNumber 필드에 지정됨
- PAN 시퀀스 번호(PSN) - PanSequenceNumber 필드에 지정됨
- 공통 세션 키(CSK)와 같은 키 파생 방법 - SessionKeyDerivationAttributes에 지정됨

- 마스터 키 파생 모드(예: EMV 옵션 A) - MajorKeyDerivationMode에서 지정됨
- 트랜잭션 데이터 - TransactionData 필드에 지정된 다양한 트랜잭션, 단말기 및 카드 데이터 문자열(예: 금액 및 날짜)
- 발급자 마스터 키 - 개별 거래를 보호하는 데 사용되는 암호화(AC) 키를 파생하는 데 사용되며 KeyIdentifier 필드에 지정된 마스터 키입니다.

주제


- 트랜잭션 데이터 구축
- 트랜잭션 데이터 패딩
- 예제


## 트랜잭션 데이터 구축

트랜잭션 데이터 필드의 정확한 내용(및 순서)은 구현 및 네트워크 체계에 따라 다르지만 최소 권장 필드(및 연결 시퀀스)는 EMV 4.4 Book 2 섹션 8.1.1 - 데이터 선택에 정의되어 있습니다. 처음 세 필드가 금액(17.00), 기타 금액(0.00), 구매 국가인 경우 트랜잭션 데이터는 다음과 같이 시작됩니다.

- 000000001700 - 금액 - 12자리(소수점 두 자리 생략)
- 000000000000 - 기타 금액 - 12자리(소수점 두 자리 생략)
- 0124 - 4자리 국가 코드
- 출력값 (부분) 트랜잭션 데이터 - 00000000170000000000000000124

## 트랜잭션 데이터 패딩

서비스로 전송하기 전에 트랜잭션 데이터를 패딩해야 합니다. 대부분의 체계에서는 ISO 9797 메서드 2 패딩을 사용하며 16진수 문자열은 필드가 암호화 블록 크기의 배수가 될 때까지 16진수 80 다음에 00이 추가됩니다. TDES의 경우 8바이트 또는 16자, AES의 경우 16바이트 또는 32자입니다. 대안(메서드 1)은 흔하지는 않지만 00만 패딩 문자로 사용합니다.

### ISO 9797 메서드 1 패딩

패딩 해제:
00000000170000000000000008400080008000084016051700000000093800000B03011203(74자 또는 37바이트)

패딩:
0000000017000000000000000840008000800008401605170000000093800000B03011203000000(80자 또는 40바이트)

### ISO 9797 메서드 2 패딩

패딩 해제:
0000000017000000000000000840008000800008401605170000000093800000B1F220103000000(80자 또는 40바이트)

패딩:
0000000017000000000000000840008000800008401605170000000093800000B1F220103000000800000(88자 또는 44바이트)

## 예제

### Visa CVN10

Example

이 예제에서는 Visa CVN10을 사용하여 생성된 ARQC를 검증합니다.

AWS Payment Cryptography가 ARQC를 검증할 수 있는 경우 http/200이 반환됩니다.
ARCQ(Authorization Request Cryptogram)가 검증되지 않으면 http/400 응답을 반환합니다.

```
$ 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 및 Visa CVN22

Example

이 예시에서는 Visa CVN18 또는 CVN22 를 사용하여 생성된 ARQC를 검증해 보겠습니다. CVN18 및 CVN22 간의 암호화 작업은 동일하지만 트랜잭션 데이터에 포함된 데이터는 다릅니다. CVN10과 비교하면 입력값이 같더라도 완전히 다른 암호가 생성됩니다.

AWS Payment Cryptography가 ARQC를 검증할 수 있는 경우 http/200이 반환됩니다. ARCQ가 검증되지 않으면 http/400을 반환합니다.

```
$ 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
 00000000170000000000000084000800080000840160517000000000093800000B1F22010300000000000
 \
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"
}
```

## HMAC 생성 및 확인

메시지 인증 코드(MAC)는 일반적으로 메시지의 무결성(수정 여부)을 인증하는 데 사용됩니다. HMAC(해시 기반 메시지 인증 코드), CBC-MAC 및 CMAC(암호 기반 메시지 인증 코드)와 같은 암호화 해시는 암호화를 활용하여 MAC 발신자를 추가로 보장합니다. HMAC는 해시 함수를 기반으로 하는 반면 CMAC는 블록 암호를 기반으로 합니다. 또한이 서비스는 CBC-MACs 유형인 ISO9797 알고리즘 1 및 3을 지원합니다.

이 서비스의 모든 HMAC 알고리즘은 암호화 해시 함수와 공유 비밀 키를 결합합니다. 키의 키 구성 요소와 같은 메시지와 비밀 키를 가져와서 고유한 태그 또는 mac를 반환합니다. 메시지의 한 문자라

도 변경되거나 비밀 키가 변경되면 결과 태그는 완전히 달라집니다. 비밀 키를 요구함으로써 암호화 HMAC는 신뢰성도 제공합니다. 비밀 키가 없다면 동일한 mac를 생성할 수 없습니다. 암호화 HMAC는 디지털 서명처럼 작동하지만 서명과 확인 모두에 단일 키를 사용하기 때문에 대칭 서명이라고도 합니다.

AWS Payment Cryptography는 다음과 같은 여러 유형의 MACs 지원합니다.

ISO9797 알고리즘 1

KeyUsage ISO9797_ALGORITHM1로 표시됩니다. 필드가 블록 크기의 배수가 아닌 경우(TDES의 경우 8바이트/16 16진수 문자, AES의 경우 16바이트/32자, AWS Payment Cryptography는 ISO9797 패딩 방법 1을 자동으로 적용합니다. 다른 패딩 방법이 필요한 경우 서비스를 호출하기 전에 적용할 수 있습니다.

ISO9797 알고리즘 3(Retail MAC)

KeyUsage ISO9797_ALGORITHM3로 표시됩니다. 알고리즘 1과 동일한 패딩 규칙이 적용됩니다.

ISO9797 알고리즘 5(CMAC)

TR31_M6_ISO_9797_5_CMAC_KEY의 KeyUsage로 표시

HMAC

TR31_M7_HMAC_KEY의 KeyUsage로 표시(HMAC_SHA224, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512 포함)

AS2805.4.1 MAC

TR31_M0_ISO_16609_MAC_KEYKeyUsage로 표시됩니다. AS2805에 대한 자세한 내용은 섹션을 참조하세요. ???

DUKPT MAC

DUKPT MAC은 일반적으로 결제 터미널과 주고받는 메시지의 소스 및 페이로드를 확인하는 데 사용됩니다. DUKPT 유도 기술을 사용하여 키를 추출한 다음 MAC를 수행합니다. 이 옵션과 함께 사용되는 키는 TR31_B0_BASE_DERIVATION_KEYKeyUsage의 로 표시됩니다.

EMV MAC

EMV MAC은 일반적으로 EMV 설명서에서 무결성 키라고 합니다. EMV 유도 기술을 사용하여 키를 도출한 다음 내부적으로 ISO9797_ALGORITHM3를 활용합니다. 일반적으로 재프로그래밍을 위해 발급자 스크립트를 칩 카드에 전송하는 데 사용됩니다. 이 옵션과 함께 사용되는 키는 TR31_E2_EMV_MKEY_INTEGRITYKeyUsage의 로 표시됩니다. 스크립트를 보내고 오프라인 핀을 업데이트하는 경우 이러한 작업을 모두 수행하는 GenerateMacEmvPinChange를 참조하세요.

주제

- MAC 생성
- MAC 검증

# MAC 생성

MAC API 생성은 알려진 암호화 키를 사용하여 전송 당사자와 수신 당사자 간의 데이터 검증을 위한 MAC(메시지 인증 코드)를 생성하여 카드 마그네틱 스트라이프에서 데이터 추적과 같은 카드 관련 데이터를 인증하는 데 사용됩니다. MAC 생성에 사용되는 데이터에는 메시지 데이터, 비밀 MAC 암호화 키 및 전송을 위한 고유한 MAC 값을 생성하는 MAC 알고리즘이 포함됩니다. MAC의 수신 당사자는 동일한 MAC 메시지 데이터, MAC 암호화 키 및 알고리즘을 사용하여 비교 및 데이터 인증을 위해 다른 MAC 값을 다시 만듭니다. 메시지의 한 문자라도 변경되거나 인증에 사용된 MAC 키가 동일하지 않은 경우 결과 MAC 값은 달라집니다. API는이 작업에 대해 ISO 9797-1 알고리즘 1 및 ISO 9797-1 알고리즘 3 MAC(정적 MAC 키 및 파생된 DUKPT 키 사용), HMAC 및 EMV MAC 암호화 키를 지원합니다.

`message-data`의 입력 값은 hexBinary 데이터여야 합니다.

이 API의 모든 옵션에 대한 자세한 내용은 GenerateMac 및 VerifyMac을 참조하세요.

선택적 파라미터 mac-length를 사용하면 출력 값을 잘라낼 수 있습니다(코드 내에서도 가능합니다). 길이 8은 8바이트 또는 16 16진수 문자를 나타냅니다.

MAC 키는 CreateKey를 호출하여 AWS Payment Cryptography로 생성하거나 ImportKey를 호출하여 가져올 수 있습니다.

> **Note**
>
> CMAC 및 HMAC 알고리즘에는 패딩이 필요하지 않습니다. 다른 모든 경우에는 TDES의 경우 8바이트(16 16진수 문자), AES의 경우 16바이트(32 16진수 문자)의 배수인 알고리즘의 블록 크기로 데이터를 패딩해야 합니다.

예제

- HMAC 생성
- ISO 9797-1 알고리즘 3을 사용하여 MAC 생성
- CMAC를 사용하여 MAC 생성
- DUKPT CMAC를 사용하여 MAC 생성

## HMAC 생성

이 예제에서는 HMAC 알고리즘 HMAC_SHA256과 HMAC 암호화 키를 사용하여 카드 데이터 인증을 위한 HMAC(해시 기반 메시지 인증 코드)를 생성합니다. 키에는 KeyUse가 TR31_M7_HMAC_KEY로 설정되고 KeyModeSofUse는 Generate로 설정되어 있어야 합니다. 해시 길이(예: 256)는 키가 생성될 때 정의되며 수정할 수 없습니다.

선택적 mac 길이 파라미터는 출력 MAC를 잘라내지만 서비스 외부에서도 수행할 수 있습니다. 이 값은 바이트 단위이므로 16의 값은 길이가 32인 16진수 문자열을 예상합니다.

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"
}
```

## ISO 9797-1 알고리즘 3을 사용하여 MAC 생성

이 예제에서는 카드 데이터 인증을 위해 ISO 9797-1 알고리즘 3(소매 MAC)을 사용하여 MAC를 생성합니다. 키에는 KeyUse가 TR31_M3_ISO_9797_3_MAC_KEY로 설정되고 KeyModeSofUse는 Generate로 설정되어 있어야 합니다.

Example

```
$ aws payment-cryptography-data generate-mac \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h \
    --message-data
 "3b3130383834393030313037333934313533d32343038323236303030373030303f33" \
    --generation-attributes="Algorithm=ISO9797_ALGORITHM3"
```

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

## CMAC를 사용하여 MAC 생성

CMAC는 키가 AES이지만 TDES도 지원하는 경우에 가장 일반적으로 사용됩니다. 이 예제에서는 AES 키를 사용한 카드 데이터 인증을 위해 CMAC(ISO 9797-1 알고리즘 5)를 사용하여 MAC를 생성합니다. 키에는 KeyUse가 TR31_M6_ISO_9797_5_CMAC_KEY로 설정되고 KeyModeSofUse는 Generate로 설정되어 있어야 합니다.

Example

```
$ aws payment-cryptography-data generate-mac \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
    --message-data
 "3b3130383834393030313037333934313533d32343038323236303030373030303f33" \
    --generation-attributes Algorithm="CMAC"
```

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

### DUKPT CMAC를 사용하여 MAC 생성

이 예제에서는 카드 데이터 인증을 위해 CMAC와 함께 DUKPT(트랜잭션당 파생된 고유 키)를 사용하여 MAC를 생성합니다. 키는 KeyUsage를 로 설정하고 `TR31_B0_BASE_DERIVATION_KEY` KeyModesOfUse를 true로 `DeriveKey` 설정해야 합니다.

DUKPT 키는 기본 파생 키(BDK)와 키 일련 번호(KSN)를 사용하여 각 트랜잭션에 대해 고유한 키를 파생합니다.

Example

```
$ aws payment-cryptography-data generate-mac --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/qnobl5lghrzunce6 --message-data
 "3b3130383834393030313037333934313533d32343038323236303030373030303f33" --generation-
attributes="DukptCmac={KeySerialNumber="932A6E954ABB32DD00000001",DukptKeyVariant=BIDIRECTIONAL
```

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

## MAC 검증

MAC API가 카드 관련 데이터 인증을 위한 MAC(메시지 인증 코드)를 확인하는 데 사용되는지 검증합니다. 인증을 위한 MAC 값을 재생성하려면 MAC 생성 시 사용한 것과 동일한 암호화 키를 사용해야 합니다. MAC 암호화 키는 CreateKey를 호출하여 AWS Payment Cryptography로 생성하거나 ImportKey를 호출하여 가져올 수 있습니다. API는이 작업에 대해 DUKPT MAC, HMAC 및 EMV MAC 암호화 키를 지원합니다.

값이 확인되면 응답 파라미터 `MacDataVerificationSuccessful`는 `Http/200`를 반환하고, 그렇지 않으면 `Mac verification failed`를 나타내는 메시지와 함께 `Http/400`를 반환합니다.

예제

- HMAC 확인
- DUKPT CMAC를 사용하여 MAC 확인

## HMAC 확인

이 예제에서는 HMAC 알고리즘 HMAC_SHA256과 HMAC 암호화 키를 사용하여 카드 데이터 인증을 위한 HMAC(해시 기반 메시지 인증 코드)를 검증합니다. 키는 KeyUsage를 로 설정하고 TR31_M7_HMAC_KEY KeyModesOfUse를 true로 Verify 설정해야 합니다.

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"
}
```

## DUKPT CMAC를 사용하여 MAC 확인

이 예제에서는 카드 데이터 인증을 위해 CMAC와 함께 DUKPT(트랜잭션당 파생된 고유 키)를 사용하여 MAC를 확인합니다. 키는 KeyUsage를 로 설정하고 TR31_B0_BASE_DERIVATION_KEY KeyModesOfUse를 true로 DeriveKey 설정해야 합니다. DUKPT 키는 기본 파생 키(BDK)와 키 일련 번호(KSN)를 사용하여 각 트랜잭션에 대해 고유한 키를 파생합니다. DukptKeyVariant의 값은 발신자와 수신자 간에 일치해야 합니다. REQUEST는 일반적으로 단일 키를 양방향으로 사용할 때 터미널에서 백엔드로, 백엔드에서 터미널로, 그리고 양방향으로 사용할 때 양방향으로 사용됩니다.

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"
}
```

# 암호화 작업을 위한 유효한 키

특정 키는 특정 작업에만 사용할 수 있습니다. 또한 일부 작업에서는 키의 키 사용 모드가 제한될 수 있습니다. 허용되는 조합은 다음 표를 참조하세요.

> **Note**
>
> 특정 조합이 허용되더라도 CVV 코드 (`generate`)를 생성한 후 이를 확인할 수 없는 (`verify`)와 같이 사용할 수 없는 상황을 만들 수 있습니다.

주제

- GenerateCardData
- VerifyCardData
- GeneratePinData(VISA/ABA 체계용)
- GeneratePinData(IBM3624용)
- VerifyPinData(비자/ABA 체계용)
- VerifyPinData(IBM3624용)
- 데이터 해독

- 데이터 암호화
- PIN 데이터 변환
- MAC 생성/확인
- GenerateMacEmvPinChange
- VerifyAuthRequestCryptogram
- 가져오기/내보내기 키
- 미사용 키 유형

## GenerateCardData

| API 엔드포인트 | 암호화 연산 또는 알고리즘 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|---|
| GenerateCardData | • AMEX_CARD_SECURITY_CODE_VERSION_1<br>• AMEX_CARD_SECURITY_CODE_VERSION_2 | TR31_C0_CARD_VERIFICATION_KEY | • TDES_2KEY<br>• TDES_3KEY | { Generate = true },{ Generate = true, Verify = true } |
| GenerateCardData | • CARD_VERIFICATION_VALUE_1<br>• 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 = true } |

| API 엔드포인트 | 암호화 연산 또는 알고리즘 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|---|
| GenerateCardData | • DYNAMIC_CARD_VERIFICATION_CODE | TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS | • TDES_2KEY | { DeriveKey = true } |
| GenerateCardData | • DYNAMIC_CARD_VERIFICATION_VALUE | TR31_E6_EMV_MKEY_OTHER | • TDES_2KEY | { DeriveKey = true } |

## VerifyCardData

| 암호화 연산 또는 알고리즘 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|
| • AMEX_CARD_SECURITY_CODE_VERSION_1<br>• AMEX_CARD_SECURITY_CODE_VERSION_2 | TR31_C0_CARD_VERIFICATION_KEY | • TDES_2KEY<br>• TDES_3KEY | { Generate = true }, { Generate = true, Verify = true } |
| • CARD_VERIFICATION_VALUE_1<br>• 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 = true } |

| 암호화 연산 또는 알고리즘 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|
| ICATION_VERIFICATION_VALUE | | | |
| • DYNAMIC_CARD_VERIFICATION_CODE | TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS | • TDES_2KEY | { DeriveKey = true } |
| • DYNAMIC_CARD_VERIFICATION_VALUE | TR31_E6_EMV_MKEY_OTHER | • TDES_2KEY | { DeriveKey = true } |

## GeneratePinData(VISA/ABA 체계용)

`VISA_PIN or VISA_PIN_VERIFICATION_VALUE`

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|
| PIN 암호화 키 | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY | • { Encrypt = true, Wrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |
| PIN 생성 키 | TR31_V2_VISA_PIN_VERIFICATION_KEY | • TDES_3KEY | • { Generate = true }<br>• { Generate = true, Verify = true } |

# GeneratePinData(**IBM3624**용)

IBM3624_PIN_OFFSET,IBM3624_NATURAL_PIN,IBM3624_RANDOM_PIN,
IBM3624_PIN_FROM_OFFSET)

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|
| PIN 암호화 키 | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY | IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET의 경우<br>• { Encrypt = true, Wrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true }<br><br>IBM 3624_PIN_OFFSET의 경우<br>• { Encrypt = true, Unwrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| PIN 생성 키 | TR31_V1_IBM3624_PIN_VERIFICATION_KEY | • TDES_3KEY | • { Generate = true }<br>• { Generate = true, Verify = true } |

## VerifyPinData(비자/ABA 체계용)

`VISA_PIN`

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| PIN 암호화 키 | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY | • { Decrypt = true, Unwrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |
| PIN 생성 키 | TR31_V2_VISA_PIN_VERIFICATION_KEY | • TDES_3KEY | • { Verify = true }<br>• { Generate = true, Verify = true } |

## VerifyPinData(**IBM3624**용)

`IBM3624_PIN_OFFSET,IBM3624_NATURAL_PIN,IBM3624_RANDOM_PIN,`
`IBM3624_PIN_FROM_OFFSET)`

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| PIN 암호화 키 | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY | IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET의 경우<br>• { Decrypt = true, Unwrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |
| PIN 확인 키 | TR31_V1_IBM3624_PIN_VERIFICATION_KEY | • TDES_3KEY | • { Verify = true }<br>• { Generate = true, Verify = true } |

## 데이터 해독

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| DUKPT | TR31_B0_BASE_DERIVATION_KEY | • TDES_2KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { DeriveKey = true }<br>• { NoRestrictions = true } |

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| EMV | TR31_E1_EMV_MKEY_CONFIDENTIALITY<br><br>TR31_E6_EMV_MKEY_OTHER | • TDES_2KEY | • { DeriveKey = true } |
| RSA | TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION | • RSA_2048<br>• RSA_3072<br>• RSA_4096 | • { Decrypt = true, Unwrap=true}<br>• {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true} |
| 대칭 키 | TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • {Decrypt = true, Unwrap=true}<br>• {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true}<br>• { NoRestrictions = true } |

## 데이터 암호화

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| DUKPT | TR31_B0_BASE_DERIVATION_KEY | • TDES_2KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { DeriveKey = true }<br>• { NoRestrictions = true } |

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|
| EMV | TR31_E1_EMV_MKEY_CONFIDENTIALITY<br>TR31_E6_EMV_MKEY_OTHER | • TDES_2KEY | • { DeriveKey = true } |
| RSA | TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION | • RSA_2048<br>• RSA_3072<br>• RSA_4096 | • { Encrypt = true, Wrap=true}<br>• {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true} |
| 대칭 키 | TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • {Encrypt = true, Wrap=true}<br>• {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true}<br>• { NoRestrictions = true } |

## PIN 데이터 변환

| Direction | 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|---|
| 인바운드 데이터 소스 | DUKPT | TR31_B0_BASE_DERIVATION_KEY | • TDES_2KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { DeriveKey = true }<br>• { NoRestrictions = true } |

| Direction | 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
|---|---|---|---|---|
| 인바운드 데이터 소스 | 비 DUKPT(PEK, AWK, IWK 등) | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Decrypt = true, Unwrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |
| 아웃바운드 데이터 대상 | DUKPT | TR31_B0_BASE_DERIVATION_KEY | • TDES_2KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { DeriveKey = true }<br>• { NoRestrictions = true } |
| 아웃바운드 데이터 대상 | 비 DUKPT(PEK, IWK, AWK 등) | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Encrypt = true, Wrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |

## MAC 생성/확인

MAC 키는 메시지/데이터 본문의 암호화 해시를 생성하는 데 사용됩니다. 매칭 작업을 수행할 수 없으므로 키 사용 모드가 제한된 키를 생성하는 것은 권장되지 않습니다. 그러나 다른 시스템이 작업 페어의 나머지 절반을 수행하려는 경우 하나의 작업으로만 키를 가져오거나 내보낼 수 있습니다.

| 허용된 키 사용 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| MAC 키 | TR31_M1_ISO_9797_1_MAC_KEY | • TDES_2KEY<br>• TDES_3KEY | • { Generate = true }<br>• { Generate = true, Verify = true }<br>• { Verify = true }<br>• { Generate = true } |
| MAC 키(소매 MAC) | TR31_M1_ISO_9797_3_MAC_KEY | • TDES_2KEY<br>• TDES_3KEY | • { Generate = true }<br>• { Generate = true, Verify = true }<br>• { Verify = true }<br>• { Generate = true } |
| MAC 키(CMAC) | TR31_M6_ISO_9797_5_CMAC_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Generate = true }<br>• { Generate = true, Verify = true }<br>• { Verify = true }<br>• { Generate = true } |
| MAC 키(HMAC) | TR31_M7_HMAC_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Generate = true }<br>• { Generate = true, Verify = true }<br>• { Verify = true } |
| MAC 키(AS2805) | TR31_M0_ISO_16609_MAC_KEY | • TDES_2KEY<br>• TDES_3KEY | • { Generate = true }<br>• { Generate = true, Verify = true }<br>• { Verify = true } |

# GenerateMacEmvPinChange

GenerateMacEmvPinChange는 EMV 오프라인 PIN 변경 작업을 위한 MAC 생성 및 PIN 암호화를 결합합니다. 이 작업에는 MAC 생성을 위한 무결성 키와 PIN 암호화를 위한 기밀성 키라는 두 가지 키 유형이 필요합니다.

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| 보안 메시징 무결성 키 | TR31_E2_EMV_MKEY_INTEGRITY | • TDES_2KEY | • { NoRestrictions = true } |
| 보안 메시징 기밀 키 | TR31_E1_EMV_MKEY_CONFIDENTIALITY | • TDES_2KEY | • { DeriveKey = true } |
| 현재 PIN PEK(PIN 암호화 키) | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Decrypt = true, Unwrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |
| 새 PIN PEK(PIN 암호화 키) | TR31_P0_PIN_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Decrypt = true, Unwrap = true }<br>• { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }<br>• { NoRestrictions = true } |

| 키 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| ARQC 키<br><br>**Note**<br>Visa 및 Amex 파생 체계에만 적용됩니다. | TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS | • TDES_2KEY | • { DeriveKey = true } |

## VerifyAuthRequestCryptogram

| 허용된 키 사용 | EMV 옵션 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| • 옵션 A<br>• 옵션 B | TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS | • TDES_2KEY | • { DeriveKey = true } |

## 가져오기/내보내기 키

| 작업 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| TR-31 래핑 키 | TR31_K1_KEY_BLOCK_PROTECTION_KEY<br><br>TR31_K0_KEY_ENCRYPTION_KEY | • TDES_2KEY<br>• TDES_3KEY<br>• AES_128<br>• AES_192<br>• AES_256 | • { Encrypt = true, Wrap = true }(내보내기 전용)<br>• { Decrypt = true, Unwrap = true }(가져오기 전용)<br>• { Encrypt = true, Decrypt = true, |

| 작업 유형 | 허용된 키 사용 | 허용된 키 알고리즘 | 허용된 키 사용 모드 조합 |
| --- | --- | --- | --- |
| | | | Wrap = true, Unwrap = true } |
| 신뢰할 수 있는 CA 가져오기 | TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE | • RSA_2048<br>• RSA_3072<br>• RSA_4096 | • { Verify = true } |
| 비대칭 암호화를 위한 퍼블릭 키 인증서 가져오기 | TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION | • RSA_2048<br>• RSA_3072<br>• RSA_4096 | • { Encrypt=true,Wrap=true } |
| ECDH와 같은 주요 계약 알고리즘에 사용되는 키 | TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT | • ECC_NIST_P256<br>• ECC_NIST_P384<br>• ECC_NIST_P521 | • { DeriveKey = true } |

## 미사용 키 유형

다음 키 유형은 현재 AWS Payment Cryptography에서 사용되지 않습니다.

- TR31_P1_PIN_GENERATION_KEY

# 일반 사용 사례

AWS Payment Cryptography는 많은 일반적인 결제 암호화 작업을 지원합니다. 다음 주제는 일반적인 사용 사례에 이러한 작업을 사용하는 방법에 대한 가이드 역할을 합니다. 모든 명령 목록은 AWS Payment Cryptography API를 검토하세요.

주제


- 발급자 및 발급자 프로세서
- 인수 및 결제 진행자


## 발급자 및 발급자 프로세서

발급자 사용 사례는 일반적으로 몇 부분으로 구성됩니다. 이 섹션은 함수(예: 핀 작업)별로 구성됩니다. 프로덕션 시스템에서 키는 일반적으로 지정된 카드 빈으로 범위가 지정되며 다음과 같이 인라인이 아닌 빈 설정 중에 생성됩니다.

주제


- 일반 함수
- 네트워크별 함수


### 일반 함수

주제


- 무작위 핀과 연결된 PVV를 생성한 다음 값을 확인합니다.
- 지정된 카드에 대한 CVV 생성 또는 확인
- 특정 카드에 대한 CVV2 생성 또는 확인
- 특정 카드에 대한 iCVV 생성 또는 확인
- EMV ARQC 확인 및 ARPC 생성
- EMV MAC 생성 및 확인
- PIN 변경을 위한 EMV MAC 생성

무작위 핀과 연결된 PVV를 생성한 다음 값을 확인합니다.

주제

- 키(들) 생성
- 무작위 핀 생성, PVV 생성, 암호화된 PIN 및 PVV 반환
- PVV 메서드를 사용하여 암호화된 PIN 검증

키(들) 생성

무작위 핀과 PVV를 생성하려면 PVV를 생성하기 위한 핀 확인 키(PVK)와 핀 암호화 키라는 두 개의 키가 필요합니다. 핀 자체는 서비스 내에서 무작위로 안전하게 생성되며 두 키와 암호화 방식으로 관련이 없습니다.

PGK는 PVV 알고리즘 자체를 기반으로 하는 알고리즘 TDES_2KEY의 키여야 합니다. PEK는 TDES_2KEY, TDES_3KEY 또는 AES_128일 수 있습니다. 이 경우 PEK는 시스템 내에서 내부적으로 사용하기 위한 것이므로 AES_128이 좋습니다. PEK가 다른 시스템(예: 카드 네트워크, 인수자, ATMs)과의 교환에 사용되거나 마이그레이션의 일부로 이동 중인 경우 TDES_2KEY가 호환성을 위해 더 적절한 선택일 수 있습니다.

PEK 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
               "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"
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

PVK 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
                "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"
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/ov6icy4ryas4zcza와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

무작위 핀 생성, PVV 생성, 암호화된 PIN 및 PVV 반환

Example

이 예제에서는 출력이 암호화된(PinData.PinBlock) 및 PIN block (pinData.VerificationValue)가 될 새 PVV (무작위) 4자리 핀을 생성합니다.PinData.PinBlock) pinData.VerificationValue). 키 입력은 PAN, Pin Verification Key(핀 생성 키라고도 함), Pin Encryption Key 및 PIN 블록 형식입니다.

이 명령을 사용하려면 키가 유형이어야 합니다TR31_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"
            }
        }
```

PVV 메서드를 사용하여 암호화된 PIN 검증

Example

이 예시에서는 지정된 PAN의 PIN을 검증해 보겠습니다. PIN은 일반적으로 카드 소지자 또는 사용자가 검증을 위해 제공하며 파일의 값과 비교됩니다(카드 소지자의 입력은 터미널 또는 기타 업스트림 공급자의 암호화된 값으로 제공됨). 이 입력을 검증하기 위해 런타임 시 다음 값 - 암호화된 핀, 입력 핀을 암호화하는 데 사용되는 키(종종 IWK라고 함) PAN 및 확인할 값( PVV 또는 )도 제공됩니다`PIN offset`.

AWS Payment Cryptography가 핀을 검증할 수 있는 경우 http/200이 반환됩니다. 핀이 검증되지 않은 경우 http/400을 반환합니다.

```
$ 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",
}
```

## 지정된 카드에 대한 CVV 생성 또는 확인

CVV 또는 CVV1은 전통적으로 카드 마그네틱 스트라이프에 포함된 값입니다. CVV2와 동일하지 않습니다(카드 소지자가 볼 수 있고 온라인 구매에 사용 가능).

첫 번째 단계는 키를 만드는 것입니다. 이 자습서에서는 CVK 이중 길이 3DES(2KEY TDES) 키를 생성합니다.

> **Note**
>
> CVV, CVV2 및 iCVV는 모두 동일한 알고리즘이 아니더라도 비슷하지만 입력 데이터를 변경합니다. 모두 동일한 키 유형 TR31_C0_CARD_VERIFICATION_KEY를 사용하지만 각 용도에 대해 별도의 키를 사용하는 것이 좋습니다. 아래 예제와 같이 별칭 및/또는 태그를 사용하여 구분할 수 있습니다.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
        }
    }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/r52o3wbqxyf6qlqr과 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

CVV 생성

Example

이 예제에서는 입력이 이고PAN, 서비스 코드(ISO/IEC 7813에서 정의)가 121이고 카드 만료 날짜가 인 지정된 PAN에 대한 CVV를 생성합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

```
$ 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"
                }
```

CVV 검증

Example

이 예제에서는 CVK, , 서비스 코드 121, 카드 만료 날짜 및 검증을 위해 트랜잭션 중에 제공된 CVV를 입력하여 지정된 PAN에 대한 CVV를 확인합니다. PAN

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

> **Note**
>
> CVV는 사용자가 입력한 값(예: CVV2)이 아니지만 일반적으로 magstripe에 포함됩니다. 제공 시 항상 검증해야 하는지 여부를 고려해야 합니다.

```
$ 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"
}
```

## 특정 카드에 대한 CVV2 생성 또는 확인

CVV2는 전통적으로 카드 뒷면에 제공되고 온라인 구매에 사용되는 값입니다. 가상 카드의 경우 앱 또는 화면에 표시될 수도 있습니다. 암호화 방식으로 CVV1과 동일하지만 서비스 코드 값이 다릅니다.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
               "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"
               }
           }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/7f7g4spf3xcklhzu와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

CVV2 생성

Example

이 예제에서는 PAN 및 카드 만료 날짜를 입력하는 지정된 PAN에 대한 CVV2를 생성합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue2를 참조하세요.

```
$ 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"
                    }
```

CVV2 검증

Example

이 예제에서는 CVK 입력, 카드 만료 날짜 PAN및 검증을 위해 트랜잭션 중에 제공된 CVV를 사용하여 지정된 PAN에 대한 CVVCVV2를 확인합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue2를 참조하세요.

> **Note**
>
> CVV2 및 기타 입력은 사용자가 입력한 값입니다. 따라서 주기적으로 검증되지 않는 문제의 징후일 필요는 없습니다.

```
$ 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"
    }
```

## 특정 카드에 대한 iCVV 생성 또는 확인

iCVV는 CVV/CVV2와 동일한 알고리즘을 사용하지만 iCVV는 칩 카드 내에 포함되어 있습니다. 서비스 코드는 999입니다.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
               "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"
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/c7dsi763r6s7lfp3과 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

iCVV 생성

Example

이 예제에서는 입력이 이고PAN, 서비스 코드(ISO/IEC 7813에서 정의)가 999이고 카드 만료 날짜가 인 지정된 PAN에 대해 iCVV를 생성합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

```
$ 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"
                    }
```

iCVV 검증

Example

검증을 위해 입력은 CVK, PAN, 999의 서비스 코드, 카드 만료 날짜 및 검증할 트랜잭션 중에 제공된 iCVV입니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

> **Note**
>
> iCVV는 사용자가 입력한 값(예: CVV2)이 아니지만 일반적으로 EMV/칩 카드에 포함됩니다. 제공 시 항상 검증해야 하는지 여부를 고려해야 합니다.

```
$ 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"
   }
```

## EMV ARQC 확인 및 ARPC 생성

ARQC(Authorization Request Cryptogram)는 EMV(칩) 카드에서 생성한 암호로, 트랜잭션 세부 정보와 승인된 카드 사용을 검증하는 데 사용됩니다. 카드, 터미널 및 트랜잭션 자체의 데이터를 통합합니다.

백엔드의 검증 시 동일한 입력이 AWS Payment Cryptography에 제공되고, 암호가 내부적으로 다시 생성되며, 트랜잭션과 함께 제공된 값과 비교됩니다. 이러한 의미에서 MAC와 유사합니다. EMV 4.4 Book 2는이 함수의 세 가지 측면, 즉 일회성 트랜잭션 키를 생성하기 위한 키 유도 방법(일반 세션 키 - CSK라고 함), 최소 페이로드 및 응답 생성 방법(ARPC)을 정의합니다.

개별 카드 체계는 통합할 추가 트랜잭션 필드 또는 해당 필드가 나타나는 순서를 지정할 수 있습니다. 다른(일반적으로 더 이상 사용되지 않는) 체계별 파생 체계도 존재하며이 설명서의 다른 부분에서 다룹니다.

자세한 내용은 API 가이드의 VerifyCardValidationData를 참조하세요.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
                "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"
                }
            }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk와 같이 키를 `KeyArn` 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

ARQC 생성

ARQC는 EMV 카드에서만 생성됩니다. 따라서 AWS Payment Cryptography에는 이러한 페이로드를 생성할 수 있는 기능이 없습니다. 테스트 목적으로 적절한 페이로드를 생성할 수 있는 여러 라이브러리와 다양한 체계에서 일반적으로 제공하는 알려진 값을 온라인으로 사용할 수 있습니다.

ARQC 검증

Example

AWS Payment Cryptography가 ARQC를 검증할 수 있는 경우 http/200이 반환됩니다. 선택적으로 ARPC(응답)를 제공하고 ARQC가 검증된 후 응답에 포함할 수 있습니다.

```
$ 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"
}
```

## EMV MAC 생성 및 확인

EMV MAC은 EMV 파생 키의 입력을 사용한 다음 결과 데이터에 대해 ISO9797-3(소매) MAC을 수행하는 MAC입니다. EMV MAC은 일반적으로 스크립트 차단 해제와 같은 명령을 EMV 카드에 전송하는 데 사용됩니다.

> **Note**
>
> AWS Payment Cryptography는 스크립트의 내용을 검증하지 않습니다. 포함할 특정 명령에 대한 자세한 내용은 스키마 또는 카드 설명서를 참조하세요.

자세한 내용은 API 가이드의 MacAlgorithmEmv를 참조하세요.

주제

- 키 생성

- EMV MAC 생성

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
        "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"
        }
    }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

EMV MAC 생성

일반적인 흐름은 백엔드 프로세스가 EMV 스크립트(예: 카드 차단 해제)를 생성하고,이 명령(특정 카드 하나에 특정한 일회용 키를 생성함)을 사용하여 서명한 다음 MAC를 반환하는 것입니다. 그런 다음 명령 + MAC이 카드에 전송되어 적용됩니다. 카드에 명령을 보내는 것은 AWS Payment Cryptography 범위를 벗어납니다.

> **Note**
>
> 이 명령은 암호화된 데이터(예: PIN)가 전송되지 않는 명령을 위한 것입니다. 이 명령을 호출하기 전에 EMV 암호화를이 명령과 결합하여 암호화된 데이터를 발급자 스크립트에 추가할 수 있습니다.

메시지 데이터

메시지 데이터에는 APDU 헤더와 명령이 포함됩니다. 이는 구현에 따라 다를 수 있지만이 예제는 차단 해제를 위한 APDU 헤더(84 24 00 00 08) 다음에 ATC(0007)와 이전 트랜잭션의 ARQC(999E57FD0F47CACE) 순입니다. 서비스는이 필드의 내용을 검증하지 않습니다.

세션 키 파생 모드

이 필드는 세션 키 생성 방법을 정의합니다. EMV_COMMON_SESSION_KEY는 일반적으로 새 구현에 사용되는 반면, EMV2000 | AMEX | MASTERCARD_SESSION_KEY | VISA도 사용할 수 있습니다.

MajorKeyDerivationMode

EMV 정의 모드 A, B 또는 C. 모드 A가 가장 일반적이며 AWS Payment Cryptography는 현재 모드 A 또는 모드 B를 지원합니다.

PAN

일반적으로 칩 필드 5A 또는 ISO8583 필드 2에서 사용할 수 있지만 카드 시스템에서 검색할 수도 있는 계정 번호입니다.

PSN

카드 시퀀스 번호입니다. 사용하지 않는 경우 00을 입력합니다.

SessionKeyDerivationValue

세션별 파생 데이터입니다. 파생 체계에 따라 필드 9F26의 마지막 ARQC(ApplicationCryptogram) 또는 9F36의 마지막 ATC일 수 있습니다.

패딩

패딩은 자동으로 적용되며 ISO/IEC 9797-1 패딩 방법 2를 사용합니다.

Example

```
$ aws payment-cryptography-data generate-mac --message-data
 842400000800079 99E57FD0F47CACE --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"
}
```

## PIN 변경을 위한 EMV MAC 생성

EMV PIN 변경에는 발급자 스크립트에 대한 MAC 생성과 EMV 칩 카드의 오프라인 PIN 변경에 대한 새 PIN 암호화라는 두 가지 작업이 결합됩니다. 이 명령은 핀이 칩 카드에 저장되는 특정 국가에서만 필요합니다(유럽 국가에서는 일반적임). 이는 카드 소지자가 PIN을 변경해야 하고 새 PIN을 MAC와 함께 카드에 안전하게 전송하여 명령의 신뢰성을 확인해야 할 때 일반적으로 사용됩니다.

> **Note**
>
> 카드에 명령을 전송하기만 하고 PIN은 변경하지 않으려면 대신 ARPC CSU 또는 EMV MAC 명령 생성을 사용하는 것이 좋습니다.

자세한 내용은 API 가이드의 GenerateMacEmvPinChange를 참조하세요.

PIN 변경을 위한 EMV MAC 및 암호화된 PIN 생성

이 작업에는 MAC 생성을 위한 EMV 무결성 키(KeyUsage: TR31_E2_EMV_MKEY_INTEGRITY)와 PIN 암호화를 위한 EMV 기밀 키(KeyUsage: TR31_E4_EMV_MKEY_CONFIDENTIALITY)의 두 가지 키가 필요합니다. 일반적인 흐름은 백엔드 프로세스가 발급자 스크립트의 MAC와 암호화된 새 PIN을 모두 포함하는 EMV PIN 변경 스크립트를 생성하는 것입니다. 그런 다음 명령과 암호화된 PIN이 카드에 전송되어 오프라인 PIN을 업데이트합니다. 카드에 명령을 보내는 것은 AWS Payment Cryptography 범위를 벗어납니다.

메시지 데이터

메시지 데이터에는 발급자 스크립트에 대한 APDU 명령이 포함됩니다. 서비스는이 필드의 내용을 검증하지 않습니다.

새로운 암호화된 PIN 블록

카드에 전송될 새 암호화된 PIN 블록입니다. PIN 암호화 키를 사용하여 암호화된 값으로 제공해야 합니다.

새 PIN PEK 식별자

이 API로 전달되기 전에 새 PIN을 암호화하는 데 사용되는 키입니다.

보안 메시징 무결성 키

MAC 생성에 사용되는 EMV 무결성 키(KeyUsage: TR31_E2_EMV_MKEY_INTEGRITY)입니다.

보안 메시징 기밀 키

PIN 암호화에 사용되는 EMV 기밀 키(KeyUsage: TR31_E4_EMV_MKEY_CONFIDENTIALITY)입니다.

MajorKeyDerivationMode

EMV는 모드 A, B 또는 C를 정의합니다. 모드 A는 가장 일반적이고 AWS Payment Cryptography는 현재 모드 A 또는 모드 B를 지원합니다.

Mode

암호화 모드, 일반적으로 PIN 변경 작업을 위한 CBC입니다.

PAN

일반적으로 칩 필드 5A 또는 ISO8583 필드 2에서 사용할 수 있지만 카드 시스템에서 검색할 수도 있는 계정 번호입니다.

PanSequenceNumber

카드 시퀀스 번호입니다. 사용하지 않는 경우 00을 입력합니다.

ApplicationCryptogram

세션별 파생 데이터로, 일반적으로 필드 9F26의 마지막 ARQC입니다.

PinBlockLengthPosition

PIN 블록 길이가 인코딩되는 위치를 지정합니다. 일반적으로 NONE으로 설정됩니다. 확실하지 않은 경우 카드 체계 사양을 확인하세요.

PinBlockPaddingType

PIN 블록의 패딩 유형을 지정합니다. 일반적으로 NO_PADDING으로 설정됩니다. 확실하지 않은 경우 카드 체계 사양을 확인하세요.

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"
}
```

# 네트워크별 함수

주제


- Visa 특정 함수
- 마스터카드 특정 함수
- American Express 특정 함수
- JCB 특정 함수


## Visa 특정 함수

주제


- ARQC - CVN18/CVN22
- ARQC - CVN10
- 3DS CAVV V7
- dCVV(동적 카드 확인 값) - CVN17


### ARQC - CVN18/CVN22

CVN18 및 CVN22는 키 파생의 CSK 메서드를 활용합니다. 정확한 트랜잭션 데이터는이 두 가지 방법에 따라 다릅니다. 트랜잭션 데이터 필드 구성에 대한 자세한 내용은 체계 설명서를 참조하세요.

### ARQC - CVN10

CVN10은 세션(트랜잭션당) 파생이 아닌 카드당 키 파생을 사용하고 다른 페이로드를 사용하는 EMV 트랜잭션에 대한 이전 Visa 메서드입니다. 페이로드 콘텐츠에 대한 자세한 내용은 스키마에 문의하십시오.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

ARQC 검증

Example

이 예제에서는 Visa CVN10을 사용하여 생성된 ARQC를 검증합니다.

AWS Payment Cryptography가 ARQC를 검증할 수 있는 경우 http/200이 반환됩니다. ARQC가 검증되지 않은 경우 http/400 응답을 반환합니다.

```
$ 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

Visa Secure(3DS) 트랜잭션의 경우 발급자 액세스 제어 서버(ACS)에서 CAVV(카드 소지자 인증 확인 값)를 생성합니다. CAVV는 카드 소지자 인증이 이루어졌다는 증거이며, 각 인증 트랜잭션에 대해 고유하고 인증 메시지에서 획득자가 제공합니다. CAVV v7은 판매자 이름, 구매 금액, 구매 날짜와 같은 요소를 포함하여 거래에 대한 추가 데이터를 승인에 바인딩합니다. 이렇게 하면 트랜잭션 페이로드의 암호화 해시가 됩니다.

암호화 방식으로 CAVV V7은 CVV 알고리즘을 사용하지만 입력이 모두 변경/사용되었습니다. CAVV V7 페이로드를 생성하기 위해 입력을 생성하는 방법은 해당 타사/Visa 설명서를 참조하세요.

#### 키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
        }
    }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/dnaeyrjgdjjtw6dk와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

CAVV V7 생성

Example

이 예제에서는 사양에 지정된 대로 입력이 있는 지정된 트랜잭션에 대해 CAVV V7을 생성합니다. 이 알고리즘의 경우 필드를 재사용/재사용할 수 있으므로 필드 레이블이 입력과 일치한다고 가정해서는 안 됩니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

```
$ 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"
            }
```

CAVV V7 검증

Example

검증의 경우 입력은 CVK, 계산된 입력 값 및 검증할 트랜잭션 중에 제공된 CAVV입니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 CardVerificationValue1을 참조하세요.

> **Note**
>
> CAVV는 사용자가 입력한 값(예: CVV2)이 아니지만 발급자 ACS에 의해 계산됩니다. 제공 시 항상 검증해야 하는지 여부를 고려해야 합니다.

```
$ 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(동적 카드 확인 값) - CVN17

dCVV(동적 카드 확인 값)는 비접촉 EMV 트랜잭션에 사용되는 Visa별 동적 암호입니다. 초기 EMV라고 하며 각 트랜잭션에 대해 고유한 확인 값을 생성하여 보안을 강화합니다. dCVV는 기본 계정 번호(PAN), PAN 시퀀스 번호(PSN), 애플리케이션 트랜잭션 카운터(ATC), 예측할 수 없는 번호 및 추적 데

이터를 포함한 입력을 사용합니다. 여전히 일부 위치에서 사용되지만 대부분 CVN18과 같은 다른 알고리즘으로 대체되었습니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 DynamicCardVerificationValue를 참조하세요.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/mw7dn3qxvkfh8ztc와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

dCVV 생성

Example

이 예제에서는 비접촉 EMV 트랜잭션에 대한 dCVV를 생성합니다. 입력에는 PAN, PAN 시퀀스 번호, 애플리케이션 트랜잭션 카운터, 예측할 수 없는 번호 및 추적 데이터가 포함됩니다.

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/mw7dn3qxvkfh8ztc \
    --primary-account-number=5111112627662122 \
    --generation-attributes
 DynamicCardVerificationValue='{ApplicationTransactionCounter=01,PanSequenceNumber=00,TrackData
 \
    --validation-data-length 5
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
mw7dn3qxvkfh8ztc",
    "KeyCheckValue": "A8E4D2",
    "ValidationData": "36667"
}
```

dCVV 검증

Example

이 예제에서는 트랜잭션 중에 제공된 dCVV를 검증합니다. 검증을 위해 생성에 사용된 것과 동일한 입력을 제공해야 합니다.

AWS Payment Cryptography를 검증할 수 있는 경우 http/200이 반환됩니다. 값이 검증되지 않으면 http/400 응답을 반환합니다.

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/mw7dn3qxvkfh8ztc \
    --primary-account-number=5111112627662122 \
    --validation-data=36667 \
    --verification-attributes
 DynamicCardVerificationValue='{ApplicationTransactionCounter=01,PanSequenceNumber=00,TrackData
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
mw7dn3qxvkfh8ztc",
    "KeyCheckValue": "A8E4D2"
}
```

## 마스터카드 특정 함수

주제

- DCVC3
- ARQC - CVN14/CVN15
- ARQC - CVN12/CVN13
- 3DS SPA2"

### DCVC3

DCVC3는 EMV CSK 및 Mastercard CVN12 체계보다 앞서 동적 키를 활용하기 위한 또 다른 접근 방식을 나타냅니다. 경우에 따라 다른 사용 사례에도 사용됩니다. 이 체계에서 입력은 PAN, PSN, Track1/Track2 데이터, 예측할 수 없는 숫자 및 트랜잭션 카운터(ATC)입니다.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
        }
    }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/hrh6qgbi3sk4y3wq와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

DCVC3 생성

Example

DCVC3는 일반적으로 칩 카드에 의해 생성되지만이 예제와 같이 수동으로 생성할 수도 있습니다.

```
$ 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"
  }
```

DCVC3 검증

Example

이 예제에서는 DCVC3를 검증합니다. ATC는 16진수로 제공되어야 합니다. 예를 들어 카운터 11은 000B로 표시되어야 합니다. 서비스에는 3자리 DCVC3가 필요하므로 4(또는 5)자리 값을 저장한 경우 3자리가 될 때까지 왼쪽 문자를 잘라내기만 하면 됩니다(예: 15321의 경우 validation-data 값이 321이어야 함).

AWS Payment Cryptography를 검증할 수 있는 경우 http/200이 반환됩니다. 값이 검증되지 않으면 http/400 응답을 반환합니다.

```
$ 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 - CVN14/CVN15

CVN14 및 CVN15는 키 파생의 EMV CSK 메서드를 활용합니다. 정확한 트랜잭션 데이터는이 두 가지 방법에 따라 다릅니다. 트랜잭션 데이터 필드 구성에 대한 자세한 내용은 체계 설명서를 참조하세요.

ARQC - CVN12/CVN13

CVN12 및 CVN13은 예측할 수 없는 숫자를 트랜잭션별 파생에 통합하고 다른 페이로드를 사용하는 EMV 트랜잭션에 대한 이전 Mastercard별 메서드입니다. 페이로드 콘텐츠에 대한 자세한 내용은 스키마에 문의하십시오.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

ARQC 검증

Example

이 예제에서는 Mastercard CVN12를 사용하여 생성된 ARQC를 검증합니다.

AWS Payment Cryptography가 ARQC를 검증할 수 있는 경우 http/200이 반환됩니다. ARQC가 검증되지 않은 경우 http/400 응답을 반환합니다.

```
$ 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
 00000000170000000000000008400080008000084016051700000000093800000B1F220103000000000000000000
 \
  --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 SPA2"

SPA2(Secure Payment Application)AAP(계정 인증 값)는 Mastercard 3DS 트랜잭션(Mastercard Idenity Check라고도 함)에 사용됩니다. HMAC 기반 MAC 생성을 사용하여 전자 상거래 트랜잭션에 대한 암호화 인증을 제공합니다. 트랜잭션별 데이터 및 공유 보안 키를 사용하여 생성되며,

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
        }
    }
```

arn:aws:payment-cryptography:us-west-2:111122223333:key/q5vjtshsg67cz5gn과 같이 키를 `KeyArn` 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

SPA2" 생성

Example

이 예제에서는 HMAC MAC 생성을 사용하여 SPA2 AAA의 발급자 인증 값(IAV) 구성 요소를 생성합니다. 메시지 데이터에는 인증할 트랜잭션별 정보가 포함되어 있습니다. 메시지 데이터의 형식은 Mastercard의 SPA2 사양을 따라야 하며이 예제에서는 다루지 않습니다.

> **Note**
>
> IAV를 AAA 값에 삽입하려면 Mastercard 사양에서 형식을 검토하세요.

```
$ 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"
}
```

SPA2AA 확인

Example

이 예제에서는 SPA2AA를 확인합니다. 확인을 위해 동일한 메시지 데이터와 MAC 값이 제공됩니다.

AWS Payment Cryptography가 MAC를 검증할 수 있는 경우 http/200이 반환됩니다. MAC가 검증되지 않은 경우 http/400 응답을 반환합니다.

```
$ 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"
}
```

## American Express 특정 함수

주제

- CSC1
- CSC2
- iCSC
- 3DS AEVV

CSC1

CSC 버전 1은 Classic CSC 알고리즘이라고도 합니다. 서비스는 3, 4 또는 5자리 숫자로 제공할 수 있습니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 AmexCardSecurityCodeVersion1을 참조하세요.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
                "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"
```

```
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/esh6hn7pxdtzgq와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

CSC1 생성

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"
  }
```

CSC1 검증

Example

이 예제에서는 CSC1을 검증합니다.

AWS Payment Cryptography를 검증할 수 있는 경우 http/200이 반환됩니다. 값이 검증되지 않으면 http/400 응답을 반환합니다.

```
$ 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 버전 2는 향상된 CSC 알고리즘이라고도 합니다. 서비스는 3, 4 또는 5자리 숫자로 제공할 수 있습니다. CSC2의 서비스 코드는 일반적으로 000입니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 AmexCardSecurityCodeVersion2를 참조하세요.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
            "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"
```

```
            }
        }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/erlm445qvunmvoda와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

CSC2 생성

이 예제에서는 길이가 4인 CSC2를 생성합니다. CSC는 3, 4 또는 5의 길이로 생성할 수 있습니다. American Express의 경우 PANs은 15자리여야 하며 34 또는 37로 시작해야 합니다. 만료 날짜는 일반적으로 YYMM 형식입니다. 서비스 코드는 다를 수 있음 - 수동을 검토하지만 일반적인 값은 000, 201 또는 702입니다.

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"
}
```

CSC2 검증

Example

이 예제에서는 CSC2를 검증합니다.

AWS Payment Cryptography를 검증할 수 있는 경우 http/200이 반환됩니다. 값이 검증되지 않으면 http/400 응답을 반환합니다.

```
$ 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

iCSC는 정적 CSC 알고리즘이라고도 하며 CSC 버전 2를 사용하여 계산됩니다. 서비스는 3, 4 또는 5 자리 숫자로 제공할 수 있습니다.

서비스 코드 999를 사용하여 연락처 카드의 iCSC를 계산합니다. 서비스 코드 702를 사용하여 비접촉 카드의 iCSC를 계산합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 AmexCardSecurityCodeVersion2를 참조하세요.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
        "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"
        }
    }
```

arn:aws:payment-cryptography:us-east-1:111122223333:key/7vrybrbjcvwtunv와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

iCSC 생성

이 예제에서는 서비스 코드 702를 사용하는 비접촉식 카드에 대해 길이가 4인 iCSC를 생성합니다. CSC는 3, 4 또는 5의 길이로 생성할 수 있습니다. American Express의 경우 PANs은 15자리여야 하며 34 또는 37로 시작해야 합니다.

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"
}
```

iCSC 검증

Example

이 예제에서는 iCSC를 검증합니다.

AWS Payment Cryptography를 검증할 수 있는 경우 http/200이 반환됩니다. 값이 검증되지 않으면 http/400 응답을 반환합니다.

```
$ 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(3-D 보안 계정 확인 값)는 American Express 3-D 보안 인증에 사용됩니다. CSC2와 동일한 알고리즘을 사용하지만 입력 파라미터는 다릅니다. 만료 날짜 필드는 예측할 수 없는(무작위) 번호로 채워야 하며, 서비스 코드는 AEVV 인증 결과 코드(1자리)와 보조 인증 코드(2자리)로 구성됩니다. 출력 길이는 3자리여야 합니다.

사용 가능한 모든 파라미터는 API 참조 가이드의 AmexCardSecurityCodeVersion2를 참조하세요.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
        "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"
        }
    }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/kw8djn5qxvh3ztm과 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

3DS AEVV 생성

이 예제에서는 길이가 3인 3DS AEVV를 생성합니다. 만료 날짜 필드에는 예측할 수 없는(임의) 번호(예: 1234)가 포함되어 있으며, 서비스 코드는 AEVV 인증 결과 코드(1자리)와 두 번째 인증 코드(2자리)로 구성됩니다. 예를 들어 543은 인증 결과 코드이고 43은 두 번째 인증 코드입니다. American Express의 경우 PANs은 15자리여야 하며 34 또는 37로 시작해야 합니다.

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"
}
```

3DS AEVV 검증

Example

이 예제에서는 3DS AEVV를 검증합니다.

AWS Payment Cryptography를 검증할 수 있는 경우 http/200이 반환됩니다. 값이 검증되지 않으면 http/400 응답을 반환합니다.

```
$ 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 특정 함수

주제

- ARQC - CVN04
- ARQC - CVN01

ARQC - CVN04

JCB CVN04는 키 파생의 CSK 메서드를 활용합니다. 트랜잭션 데이터 필드 구성에 대한 자세한 내용은 스키마 설명서를 참조하세요.

ARQC - CVN01

CVN01은 세션(트랜잭션당) 추출 대신 카드 키별 추출을 사용하고 다른 페이로드를 사용하는 EMV 트랜잭션에 대한 이전 JCB 메서드입니다. 이 메시지는 Visa에서도 사용되므로 요소 이름에는 JCB에도 사용되더라도 해당 이름이 있습니다. 페이로드 콘텐츠에 대한 자세한 내용은 체계 설명서를 참조하십시오.

키 생성

```
$ 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"}]'
```

응답은 후속 호출을 위한 ARN과 키 검사 값(KCV)을 포함한 요청 파라미터를 다시 반영합니다.

```
{
                        "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"
                        }
                    }
```

arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk와 같이 키를 KeyArn 나타내는를 기록해 둡니다. 다음 단계에서 이 작업을 수행합니다.

ARQC 검증

Example

이 예제에서는 JCB CVN01을 사용하여 생성된 ARQC를 검증합니다. 이렇게 하면 Visa 메서드와 동일한 옵션이 사용되므로 파라미터의 이름이 사용됩니다.

AWS Payment Cryptography가 ARQC를 검증할 수 있는 경우 http/200이 반환됩니다. ARQC가 검증되지 않은 경우 http/400 응답을 반환합니다.

```
$ 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"
              }
```

# 인수 및 결제 진행자

전표 매입사, PSPs 및 결제 진행자는 일반적으로 발급자와 다른 암호화 요구 사항 세트를 갖습니다. 일반적인 사용 사례는 다음과 같습니다.

데이터 복호화

데이터(특히 팬 데이터)는 결제 터미널에서 암호화할 수 있으며 백엔드에서 복호화해야 합니다. 데이터 복호화 및 데이터 암호화는 TDES, AES 및 DUKPT 파생 기술을 비롯한 다양한 방법을 지원합니다. AWS Payment Cryptography 서비스 자체도 PCI P2PE를 준수하며 PCI P2PE 복호화 구성 요소로 등록됩니다.

TranslatePin

PCI PIN 규정 준수를 유지하기 위해 획득 시스템에는 보안 디바이스에 카드 소지자 핀을 입력한 후 카드 소지자 핀이 명확하게 없어야 합니다. 따라서 터미널에서 다운스트림 시스템(예: 결제 네트워

크 또는 발급자)으로 핀을 전달하려면 결제 터미널에서 사용한 키와 다른 키를 사용하여 핀을 다시 암호화해야 합니다. Pin 번역은 암호화된 핀을 servicebbb를 사용하여 한 키에서 다른 키로 안전하게 변환하여 이를 달성합니다. 이 명령을 사용하면 TDES, AES 및 DUKPT 유도와 같은 다양한 체계와 ISO-0, ISO-3 및 ISO-4와 같은 핀 블록 형식 간에 핀을 변환할 수 있습니다.

VerifyMac

데이터가 전송 중에 수정되지 않도록 결제 터미널의 데이터를 MAC로 지정할 수 있습니다. Mac 및 GenerateMac이 ISO-9797-1 알고리즘 1, ISO-9797-1 알고리즘 ISO-9797-13(소매 MAC) 및 CMAC 기법과 함께 사용할 수 있는 TDES, AES 및 DUKPT 파생 기법을 비롯한 대칭 키를 사용하는 다양한 기법을 지원하는지 확인합니다.

추가 주제

- 동적 키 사용

## 동적 키 사용

동적 키를 사용하면와 같은 암호화 작업에 일회성 또는 제한된 사용 키를 사용할 수 있습니다`EncryptData`. 이 흐름은 키 구성 요소가 자주 교체되고(예: 모든 카드 트랜잭션에서) 키 구성 요소를 서비스로 가져오지 않으려는 경우에 사용할 수 있습니다. 수명이 짧은 키는 softPOS/Mpoc 또는 기타 솔루션의 일부로 사용할 수 있습니다.

> **Note**
>
> 이는 AWS Payment Cryptography를 사용하는 일반적인 흐름 대신 사용할 수 있습니다. 여기서 암호화 키는 생성되거나 서비스로 가져오고 키는 키 별칭 또는 키 ARN을 사용하여 지정됩니다.

다음 작업은 동적 키를 지원합니다.

- EncryptData
- DecryptData
- ReEncryptData
- TranslatePin

## 데이터 복호화

다음 예제에서는 복호화 명령과 함께 동적 키를 사용하는 방법을 보여줍니다. 이 경우 키 식별자는 복호화 키(TR-31 형식의 래핑된 키 파라미터에 제공됨)를 보호하는 래핑 키(KEK)입니다. 래핑된 키는 B 또는 D 사용 모드와 함께 복호화 명령과 함께 사용할 D0의 주요 목적이어야 합니다.

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"
}
```

## 핀 번역

다음 예제에서는 동적 키를 변환 핀 명령과 함께 사용하여 동적 키에서 반정적 획득자 작업 키(AWK)로 변환하는 방법을 보여줍니다. 이 경우 수신 키 식별자는 TR-31 형식으로 제공되는 동적 핀 암호화 키(PEK)를 보호하는 래핑 키(KEK)입니다. 래핑된 키는 B 또는 D의 사용 모드와 P0 함께의 주요 목적이어야 합니다. 발신 키 식별자는 유형의 키TR31_P0_PIN_ENCRYPTION_KEY와 Encrypt=true,Wrap=true의 사용 모드입니다.

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"
}
```

# AWS Payment Cryptography의 리전별 기능

특정 기능은 리전별로 다를 수 있으며 달리 사용되지 않을 수 있습니다. 이러한 기능은이 단원에서 자세히 설명합니다.

## AS2805

Australia Standard 2805(AS2805)는 주로 카드 기반 결제 거래에 사용되는 전자 송금의 표준입니다. 이는 Standards Australia에서 유지 관리합니다. 표준은 메시지 형식부터 암호화 표준까지 다양한 주제를 다루는 6개의 책으로 구성됩니다.

6부에서는 host-to-host(node-to-node) 통신 및 관련 암호화 요구 사항을 비롯한 키 관리에 대한 지침을 제공하는 반면, 다른 측면은 다른 부분에서 다룹니다. 이 표준의 모든 암호화는 현재 TDES를 기반으로 합니다.

> **Note**
>
> AS2805는 현재 ap-southeast-2 리전에서 사용할 수 있습니다. 가까운 시일 내에 추가 리전으로 롤아웃될 예정입니다.

AS2805는 아래에 요약된 다른 구현과 비교하여 여러 가지 차이점이 있습니다.

키 보호

TR-31/X9.143과 같은 키 블록 대신 키 변형을 사용합니다. AWS 결제 암호화는 모든 키를 키 블록으로 내부적으로 저장하지만 AS2805 정의 변형을 사용하여 가져오기, 내보내기 및 계산을 허용합니다.

단방향 키

AS2805에서는 단방향 키를 사용해야 합니다. 두 노드 모두 메시지 인증 코드(MAC)를 생성해야 하는 경우 두 개의 키를 사용합니다.

핀 블록

AS2805는 트랜잭션당 고유한 핀 암호화 키에 대한 키 유도 기술을 정의합니다. 이는 DUKPT 대신 사용할 수 있습니다. AS2805 체계는 DUKPT의 트랜잭션 카운터 사용과 비교하여 트랜잭션 데이터(트레이스 번호 및 트랜잭션 금액)에 의존합니다.

키 교환 검증

핀 키와 같은 작업 키 교환을 시작하기 전에 KEK를 검증하는 프로세스를 정의합니다. 다른 체계에서는 KEK가 자주 교환되지 않으며 KCV를 사용하여 검증됩니다.

AS2805는 키 블록이 아닌 키 변형의 개념을 사용하여 키가 의도한(및 유일한) 용도로만 사용되도록 합니다. 다음은 키를 사용하여 다른 암호화 함수를 가져오거나 내보내거나 수행할 때 AWS Payment Cryptography가 변형과 키 블록을 매핑하는 방법입니다.

| AS2805 키 유형 | AWS Payment Cryptography 키 유형 |
|---|---|
| TERMINAL_MAJOR_KEY_VARIANT_00 | TR31_K0_KEY_ENCRYPTION_KEY |
| PIN_ENCRYPTION_KEY_VARIANT_28 | TR31_P0_PIN_ENCRYPTION_KEY |
| MESSAGE_AUTHENTICATION_KEY_VARIANT_24 | TR31_M0_ISO_16609_MAC_KEY |
| DATA_ENCRYPTION_KEY_VARIANT_22 | TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY |
| VARIANT_MASK_82,VARIANT_MASK_82C0 | KEK 검증 프로세스의 일부로 사용할 수 있는 옵션입니다. 이러한 키 유형은 임시적이며 서비스에 저장되지 않습니다. |

node1과 node2라는 두 개의 노드가 주어지면 다음 예제는 node1의 관점에서 가져온 것입니다. AWS Payment Cryptography는 프로세스의 양쪽에서 APIs를 지원합니다.

주제

- 초기 키(KEK) 교환
- KEK 검증
- 작업 키 생성 및 전송
- 작업 키 내보내기
- 핀 번역
- Mac 생성 및 검증

# 초기 키(KEK) 교환

AS2805에서 각 측에는 자체 KEK가 있습니다. KEK(s)는 전송 측에서 키를 보호/래핑하고 node2로 전송해야 할 때마다 사용되는 전송 측 키를 나타냅니다. KEK(r)는 반대(node2) 측에서 생성한 키입니다.

> **Note**
>
> 이러한 용어는 상대적입니다. 한 면은 키(발신 면)를 생성하고 다른 면은 키를 수신합니다. 따라서 KEY1이 주어지면 node1에서는 KEK(s)라고 하고 node2에서는 KEK(r)라고 합니다.

KEK for AS2805는 키 블록이 아닌 암호를 보호하는 데 사용되므로 항상 키 유형 = TR31_K0_KEY_ENCRYPTION_KEY입니다. AS2805 6.1에 정의된 대로 TERMINAL_MAJOR_KEY_VARIANT_00에 매핑됩니다.

단계:

1. 키 생성

   CreateKey API를 사용하여 키를 생성합니다. TR31_K0_KEY_ENCRYPTION_KEY 유형의 키를 생성합니다.

2. node2와 키를 교환하는 방법 결정

   상대방과 KEK를 교환하는 방법을 결정합니다. AS2805의 경우 가장 일반적이고 상호 운용 가능한 방법은 RSA 래핑입니다.

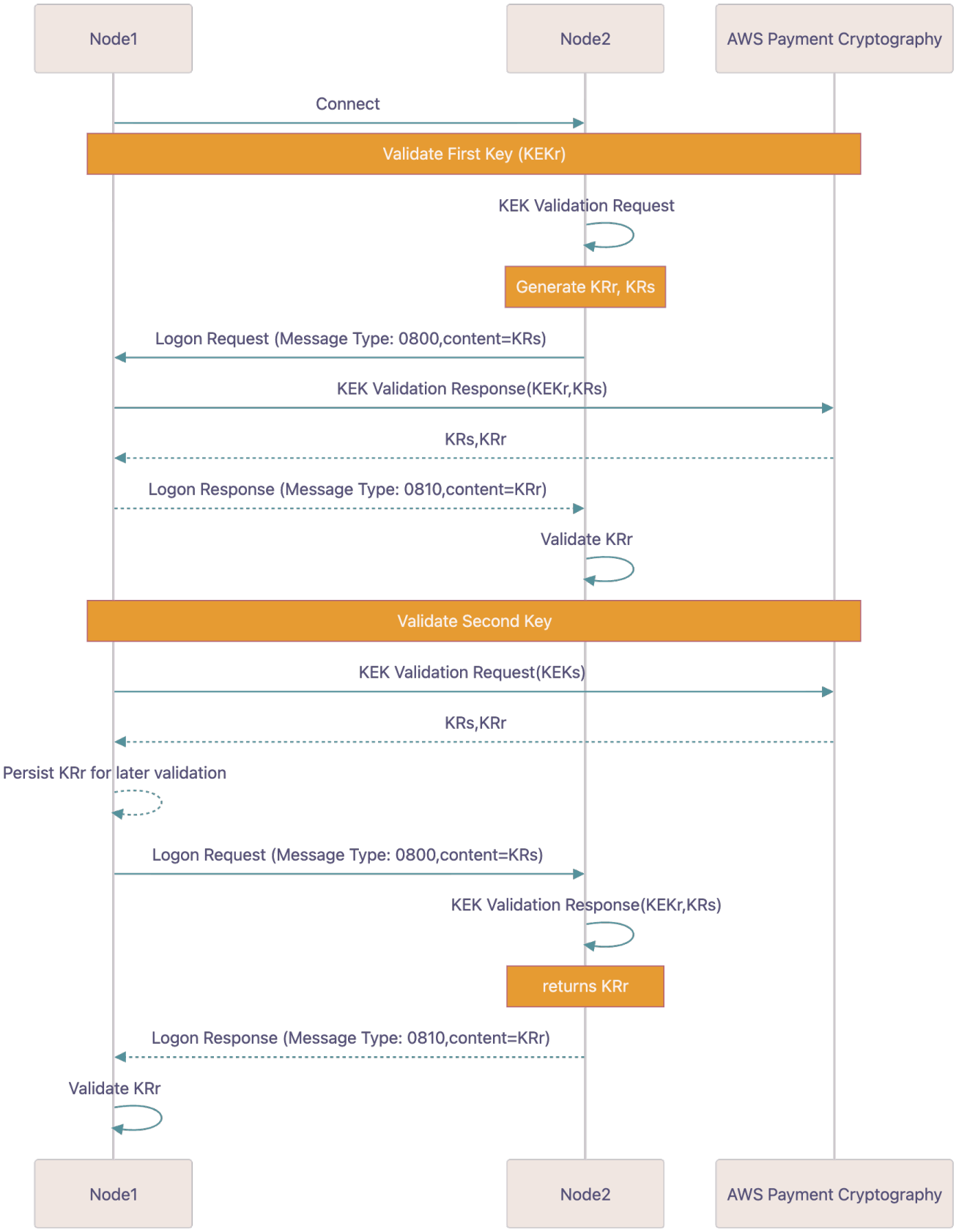
3.KEKs 내보내기

   위의 선택에 따라 node2에서 퍼블릭 키 인증서를 받게 됩니다. 해당 인증서를 사용하여 내보내기를 실행하여 키를 보호합니다(또는 ECDH를 사용하는 경우 키를 추출합니다).

4. KEKr 가져오기

   위의 선택에 따라 퍼블릭 키 인증서를 node2로 보냅니다. 해당 인증서를 사용하여 로 가져오기를 실행하여 노드 2의 KEKr를 서비스에 로드합니다.

# KEK 검증


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

서비스(node1)가 node2에 연결되면 각 측은 KEK 검증이라는 프로세스를 사용하여 후속 작업에 동일한 KEK를 사용하고 있는지 확인합니다.

1: 첫 번째 키를 검증하는 단계

1.1 KRs 수신

Node2는 KRs을 생성하여 로그온 프로세스의 일부로 사용자에게 전송합니다. AWS Payment Cryptography를 사용하여이 값 또는 다른 솔루션을 생성할 수 있습니다.

1.2 KEK 검증 응답 생성

노드는 입력이 1단계에서 제공된 KEK(r) 및 KRs인 KEK 검증 응답을 생성합니다.

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 계산된 KRr 반환

계산된 KRr을 node2로 반환합니다. 이 노드는 이를 1단계에서 계산된 값과 비교합니다.

2. 두 번째 키를 검증하는 단계

2.1 KRr 및 KRs 생성

노드는 AWS Payment Cryptography를 사용하여 무작위 값과이 값의 반전(역방향) 복사본을 생성합니다. 서비스는 KEK(들)로 래핑된 이러한 두 값을 모두 출력합니다. 이를 KR(s) 및 KR(r)이라고 합니다.

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 KRs node2로 전송

KRs node2로 전송합니다. 이후 검증을 위해 KRr을 유지합니다.

2.3 Node2에서 KEK 검증 응답 생성

Node2는 KEKr 및 KRs 사용하고 KRr을 생성하여 서비스로 다시 보냅니다.

2.4 응답 검증

1단계의 KRr과 3단계에서 반환된 값을 비교합니다. 일치하는 경우 계속 진행합니다.

# 작업 키 생성 및 전송

AS2805에 사용되는 일반적인 작업 키에는 두 가지 키 세트가 포함됩니다.

영역 핀 키(ZPK), 영역 암호화 키(ZEK) 및 영역 인증 키(ZAK)와 같은 노드 간 키입니다.

DUKPT를 사용하지 않는 경우 터미널 메인 키(TMK) 및 터미널 핀 키(TPK)와 같은 터미널과 노드 간의 키입니다.

> **Note**
>
> 터미널 키당 키를 최소화하고 가능한 경우 더 적은 수의 키를 사용하는 TR-34 및 DUKPT와 같은 기술을 활용하는 것이 좋습니다.

Example

이 예제에서는 선택적 태그를 사용하여이 키의 용도와 사용을 추적했습니다. 태그는 시스템 암호화 함수의 일부로 사용되지 않지만 분류, 재무 추적에 사용할 수 있으며 IAM 정책을 적용하는 데 사용할 수 있습니다.

```
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-data 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"
 }
}
```

## 작업 키 내보내기

다른 당사자와의 호환성을 유지하기 위해 AWS Payment Cryptography는 TR-31과 같은 키 블록 대신 키 변형을 사용하는 AS2805 대칭 키 래핑 기술을 지원합니다. 당사자 간에 여러 키를 공유하는 경우 각 키를 개별적으로 내보내야 합니다. 데이터가 양방향으로 전송되는 경우 각 측에서 메시지 인증 코드를 생성하는 데 사용하는 ZAK(s) 및 ZAK(r)와 같이 동일한 유형의 당사자 간에 두 개의 키가 있을 수 있습니다.

이러한 형식으로 가져오고 내보낼 추가 파라미터는 명령에 지정됩니다.

```
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-data 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"
    }
}
```

## 핀 번역

AS2805는 섹션 6.4의 세션별 키 유도 모드를 설명합니다. DUKPT와 같은 목적을 제공하며 DUKPT가 섹션 6.7에서 다루기 때문에 두 알고리즘 중 하나를 사용할 수 있습니다. 이 체계에서 세션 핀 키(KPE

라고 함)는 SystemTraceAuditNumber(STAN) 및 TransactionAmount를 파생 데이터로 사용하여 터미널 핀 키에서 파생됩니다.

번역 핀은 다양한 형식으로 변환할 수 있는 일반적인 함수입니다. 이 예제에서는 결제 네트워크에 핀을 보낼 때와 같이 KPE의 핀을 핀 암호화 키(PEK)로 변환합니다.

```
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 생성 및 검증

MAC 생성 및 확인 명령은 HMAC, CMAC, EMV MACs 등을 포함한 다양한 MAC를 지원합니다. AS2805의 경우 AS2805.4.1에 정의된 추가 변형이 있습니다. 일반적으로 AS2805에서 수신 메시지는 이 MAC를 사용하여 확인되며 발신 메시지에는 MAC도 포함됩니다.

```
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"
}
```

# AWS Payment Cryptography의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 -AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다.  AWS Payment Cryptography에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스규정 준수 프로그램 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 주제는 AWS Payment Cryptography를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 보안 및 규정 준수 목적에 맞게 AWS Payment Cryptography를 구성하는 방법을 보여줍니다. 또한 AWS Payment Cryptography 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

주제


- AWS Payment Cryptography의 데이터 보호
- AWS Payment Cryptography의 복원성
- 의 인프라 보안 AWS Payment Cryptography
- VPC 엔드포인트를 통해 AWS Payment Cryptography에 연결
- 하이브리드 포스트 양자 TLS 사용
- AWS Payment Cryptography의 보안 모범 사례


## AWS Payment Cryptography의 데이터 보호

AWS 공동 책임 모델 AWS Payment Cryptography의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보

안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 일반 데이터 보호 규정(GDPR) 센터를 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 AWS Payment Cryptography 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

AWS Payment Cryptography는 Payment Cryptography 키를 저장하고 보호하여 높은 가용성을 제공하는 동시에 강력하고 유연한 액세스 제어를 제공합니다.

주제

- 키 구성 요소 보호
- 데이터 암호화
- 저장 시 암호화

- 전송 중 암호화
- 인터네트워크 트래픽 개인 정보

## 키 구성 요소 보호

기본적으로 AWS Payment Cryptography는 서비스에서 관리하는 결제 키의 암호화 키 자료를 보호합니다. 또한, AWS Payment Cryptography는 서비스 외부에서 생성된 주요 자료를 가져오는 옵션을 제공합니다. 결제 키 및 키 구성 요소에 대한 기술 세부 정보는 AWS Payment Cryptography 세부 정보를 참조하세요.

## 데이터 암호화

AWS Payment Cryptography의 데이터는 AWS Payment Cryptography 키, 해당 키가 나타내는 암호화 키 구성 요소, 사용 특성으로 구성됩니다. 이 키 구성 요소는 AWS Payment Cryptography 하드웨어 보안 모듈(HSM) 내에서만 일반 텍스트로 존재하며 사용 중일 때만 존재합니다. 그렇지 않으면 키 구성 요소와 특성이 암호화되어 내구성이 뛰어난 영구 스토리지에 저장됩니다.

AWS Payment Cryptography가 결제 키에 대해 생성하거나 로드하는 키 구성 요소는 암호화되지 않은 상태로 두지 않습니다. AWS Payment Cryptography API 작업을 통해 암호화된 상태로 내보낼 수 있습니다.

## 저장 시 암호화

AWS Payment Cryptography는 PCI PTS HSM 등재 HSM에서 결제 키용 키 구성 요소를 생성합니다. 사용하지 않을 경우 키 구성 요소는 HSM 키에 의해 암호화되어 내구성이 뛰어난 영구 스토리지에 기록됩니다. Payment Cryptography 키의 키 구성 요소와 키 구성 요소를 보호하는 암호화 키는 HSM을 일반 텍스트 형식으로 남기지 않습니다.

Payment Cryptography 키에 대한 키 구성 요소의 암호화 및 관리는 전적으로 서비스에서 처리됩니다.

자세한 내용은 AWS 키 관리 서비스 암호화 세부 정보를 참조하세요.

## 전송 중 암호화

AWS Payment Cryptography가 결제 키에 대해 생성하거나 로드하는 키 구성 요소는 AWS Payment Cryptography API 작업에서 일반 텍스트로 내보내거나 전송하지 않습니다. AWS Payment Cryptography는 키 식별자를 사용하여 API 작업의 키를 나타냅니다.

그러나 일부 API 작업은 이전에 공유되거나 비대칭인 키 교환 키로 암호화된 키를 내보냅니다. 또한 고객은 API 작업을 사용하여 결제 키에 대한 암호화된 키 구성 요소를 가져올 수 있습니다.

모든 AWS Payment Cryptography API 호출은 서명되어야 하며 전송 계층 보안(TLS)을 사용하여 전송되어야 합니다. AWS Payment Cryptography에는 PCI에서 "강력한 암호화"로 정의한 TLS 버전 및 암호 제품군이 필요합니다. 모든 서비스 엔드포인트는 TLS 1.2~1.3 및 하이브리드 포스트 양자 TLS를 지원합니다.

자세한 내용은 AWS 키 관리 서비스 암호화 세부 정보를 참조하세요.

## 인터네트워크 트래픽 개인 정보

AWS Payment Cryptography는 AWS Management Console과 결제 키를 생성 및 관리하고 암호화 작업에 사용할 수 있는 API 작업 세트를 지원합니다.

AWS Payment Cryptography는 프라이빗 네트워크에서 AWS로의 두 가지 네트워크 연결 옵션을 지원합니다.

- 인터넷을 통한 IPSec VPN 연결.
- AWS Direct Connect는 표준 이더넷 광섬유 케이블을 통해 내부 네트워크를 AWS Direct Connect 위치에 연결합니다.

모든 Payment Cryptography API 직접 호출은 서명되어야 하며 전송 계층 보안(TLS)을 사용하여 전송되어야 합니다. 호출에는 또한 완벽한 순방향 보안을 지원하는 최신 암호 제품군도 필요합니다. 결제 키에 대한 키 구성 요소를 저장하는 하드웨어 보안 모듈(HSM)에 대한 트래픽은 AWS 내부 네트워크를 통해 알려진 AWS Payment Cryptography API 호스트에서만 허용됩니다.

공용 인터넷을 통해 트래픽을 전송하지 않고 Virtual Private Cloud(VPC)에서 AWS Payment Cryptography에 직접 연결하려면 AWS PrivateLink에서 제공하는 VPC 엔드포인트를 사용하세요. 자세한 내용은 VPC 엔드포인트를 통한 AWS Payment Cryptography 연결을 참조하세요.

AWS Payment Cryptography에서는 전송 계층 보안(TLS) 네트워크 암호화 프로토콜에 대한 하이브리드 포스트 퀀텀 키 교환 옵션을 지원합니다. AWS Payment Cryptography API 엔드포인트에 연결할 때 TLS와 함께 이 옵션을 사용할 수 있습니다.

# AWS Payment Cryptography의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며, 이러한 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크를 통해 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

## 리전별 격리

AWS Payment Cryptography는 여러 리전에서 사용할 수 있는 리전 서비스입니다.

지역별로 격리된 AWS Payment Cryptography 설계는 한 AWS 리전의 가용성 문제가 다른 지역의 AWS Payment Cryptography 운영에 영향을 미치지 않도록 보장합니다. AWS Payment Cryptography는 모든 소프트웨어 업데이트 및 조정 작업이 원활하고 눈에 띄지 않게 수행되므로 계획된 가동 중지 시간이 전혀 발생하지 않도록 설계되었습니다.

AWS Payment Cryptography 서비스 수준에 관한 계약(SLA)에는 모든 Payment Cryptography API에 대한 99.99%의 서비스 보장이 포함됩니다. 이 보장을 이행하기 위해 AWS Payment Cryptography는 API 요청을 실행하는 데 필요한 모든 데이터 및 권한 부여 정보가 요청을 수신하는 모든 리전 호스트에서 사용할 수 있도록 합니다.

이 AWS Payment Cryptography 인프라는 각 리전에서 최소 3개의 가용 영역(AZ)에 복제됩니다. 여러 호스트 장애가 AWS Payment Cryptography 성능에 영향을 미치지 않도록 하기 위해 한 리전의 모든 AZ에서 고객 트래픽을 처리하도록 AWS Payment Cryptography가 설계되었습니다.

결제 키의 속성 또는 권한에 대한 변경 사항은 리전의 모든 호스트에서 후속 요청을 올바르게 처리할 수 있도록 리전의 모든 호스트에 복제됩니다. 결제 키를 사용한 암호화 작업에 대한 요청은 결제 키로 작업을 수행할 수 있는 AWS Payment Cryptography 하드웨어 보안 모듈(HSM)의 플릿에 전달됩니다.

## 멀티 테넌트 디자인

AWS Payment Cryptography의 다중 테넌트 설계를 통해 가용성 SLA를 충족하고 높은 요청률을 유지하면서 키 및 데이터의 기밀성을 보호할 수 있습니다.

암호화 작업에 지정한 결제 키가 항상 사용되는 키인지 확인하기 위해 여러 무결성 보장 메커니즘이 배포됩니다.

Payment Cryptography 키의 일반 텍스트 키 구성 요소는 광범위하게 보호됩니다. 키 구성 요소는 생성되는 즉시 HSM에서 암호화되며 암호화된 키 구성 요소는 즉시 안전한 스토리지로 옮겨집니다. 암호화된 키는 사용 시 HSM 내에서 검색 및 해독됩니다. 일반 텍스트 키는 암호화 작업을 완료하는 데 필요한 시간 동안만 HSM 메모리에 남아 있습니다. 일반 텍스트 키 구성 요소는 HSM을 절대 떠나지 않으며 영구 스토리지에 절대 기록되지 않습니다.

AWS Payment Cryptography에서 키를 보호하기 위해 사용하는 메커니즘에 대한 자세한 내용은 AWS Payment Cryptography 세부 정보를 참조하세요.

# 의 인프라 보안 AWS Payment Cryptography

관리형 서비스인는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 AWS Payment Cryptography 보호됩니다.

AWS 에서 게시한 API 호출을 사용하여 네트워크를 AWS Payment Cryptography 통해에 액세스 합니다. 클라이언트가 전송 계층 보안(TLS) 1.2 이상을 지원해야 합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 Perfect Forward Secrecy(PFS)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

## 물리적 호스트의 격리

AWS Payment Cryptography가 사용하는 물리적 인프라의 보안은 Amazon Web Services: 보안 프로세스 개요의 물리적 및 환경적 보안 섹션에 설명된 제어의 적용을 받습니다. 이전 섹션에 나열된 규정 준수 보고서와 타사 감사 결과에 대한 자세한 내용을 확인할 수 있습니다.

AWS Payment Cryptography는 상용 전용 PCI PTS HSM 등재 하드웨어 보안 모듈(HSM)을 통해 지원됩니다. AWS Payment Cryptography 키에 대한 키 구성 요소는 HSM의 휘발성 메모리에만 저장되며 Payment Cryptography 키가 사용되는 동안에만 저장됩니다. HSM은 모든 물리적 액세스에 대해 이중 제어를 적용하는 Amazon 데이터 센터 내의 액세스 제어 랙에 있습니다. AWS Payment Cryptography HSM의 작동에 대한 자세한 내용은 AWS Payment Cryptography 세부 정보를 참조하세요.

# VPC 엔드포인트를 통해 AWS Payment Cryptography에 연결

Virtual Private Cloud(VPC)의 프라이빗 인터페이스 엔드포인트를 통해 AWS Payment Cryptography에 직접 연결할 수 있습니다. 인터페이스 VPC 엔드포인트를 사용하는 경우 VPC와 AWS Payment Cryptography 간의 통신은 전적으로 AWS 네트워크 내에서 수행됩니다.

AWS Payment Cryptography는 로 구동되는 Amazon Virtual Private Cloud(VPC) 엔드포인트를 지원합니다AWS PrivateLink. 각 VPC 엔드포인트는 하나 이상의 탄력적 네트워크 인터페이스(ENI) 및 VPC 서브넷의 프라이빗 IP 주소로 표현됩니다.

인터페이스 VPC 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 VPC를 AWS Payment Cryptography에 직접 연결합니다. VPC의 인스턴스는 AWS Payment Cryptography와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

리전

AWS Payment Cryptography는 AWS Payment Cryptography가 지원되는 모든 AWS 리전 에서 VPC 엔드포인트 및 VPC 엔드포인트 정책을 지원합니다.

주제

- AWS Payment Cryptography VPC 엔드포인트에 대한 고려 사항
- AWS Payment Cryptography용 VPC 엔드포인트 생성
- AWS Payment Cryptography VPC 엔드포인트에 연결
- VPC 엔드포인트에 대한 액세스 제어
- 정책 설명에 VPC 엔드포인트 사용
- VPC 엔드포인트 로깅

## AWS Payment Cryptography VPC 엔드포인트에 대한 고려 사항

> **Note**
>
> VPC 엔드포인트를 사용하면 최소 하나의 가용 영역(AZ)에서 서비스에 연결할 수 있지만 고가용성 및 중복성을 위해 세 개의 가용 영역에 연결하는 것이 좋습니다.

AWS Payment Cryptography용 인터페이스 VPC 엔드포인트를 설정하기 전에 AWS PrivateLink 가이드의 인터페이스 엔드포인트 속성 및 제한 주제를 검토하세요.

AWS VPC 엔드포인트에 대한 Payment Cryptography 지원에는 다음이 포함됩니다.

- VPC 엔드포인트를 사용하여 VPC에서 모든 AWS Payment Cryptography Control 플레인 작업과 AWS Payment Cryptography Data 플레인 작업을 호출할 수 있습니다.
- AWS Payment Cryptography 리전 엔드포인트에 연결하는 인터페이스 VPC 엔드포인트를 생성할 수 있습니다.
- AWS Payment Cryptography는 컨트롤 플레인과 데이터 플레인으로 구성됩니다. 하나 또는 두 하위 서비스를 모두 설정하도록 선택할 수 AWS PrivateLink 있지만 각 하위 서비스는 별도로 구성됩니다.
- AWS CloudTrail 로그를 사용하여 VPC 엔드포인트를 통해 AWS Payment Cryptography 키 사용을 감사할 수 있습니다. 자세한 내용은 VPC 엔드포인트 로깅을 참조하세요.

## AWS Payment Cryptography용 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 Amazon VPC API를 사용하여 AWS Payment Cryptography용 VPC 엔드포인트를 생성할 수 있습니다. 자세한 내용은 AWS PrivateLink 안내서의 인터페이스 엔드포인트 생성을 참조하세요.

- AWS Payment Cryptography용 VPC 엔드포인트를 생성하려면 다음 서비스 이름을 사용합니다.

```
com.amazonaws.region.payment-cryptography.controlplane
```

```
com.amazonaws.region.payment-cryptography.dataplane
```

  예를 들어 미국 서부(오레곤) 리전()에서 us-west-2서비스 이름은 다음과 같습니다.

```
com.amazonaws.us-west-2.payment-cryptography.controlplane
```

```
com.amazonaws.us-west-2.payment-cryptography.dataplane
```

VPC 엔드포인트를 더 쉽게 사용하려면 VPC 엔드포인트에 프라이빗 DNS 이름을 사용하도록 설정합니다. DNS 이름 활성화 옵션을 선택하면 표준 AWS Payment Cryptography DNS 호스트 이름이 VPC 엔드포인트로 확인됩니다. 예를 들어 https://controlplane.payment-cryptography.us-west-2.amazonaws.com은 서비스 이름 com.amazonaws.us-west-2.payment-cryptography.controlplane에 연결된 VPC 엔드포인트로 확인됩니다.

이 옵션을 선택하면 VPC 엔드포인트를 더 쉽게 사용할 수 있습니다. AWS SDKs는 기본적으로 표준 AWS Payment Cryptography DNS 호스트 이름을 AWS CLI 사용하므로 애플리케이션 및 명령에서 VPC 엔드포인트 URL을 지정할 필요가 없습니다.

자세한 내용은 AWS PrivateLink 가이드의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하세요.

## AWS Payment Cryptography VPC 엔드포인트에 연결

AWS SDK, AWS CLI 또는를 사용하여 VPC 엔드포인트를 통해 AWS Payment Cryptography에 연결할 수 있습니다 AWS Tools for PowerShell. VPC 엔드포인트를 지정하려면 해당 DNS 이름을 사용합니다.

예를 들어 이 list-keys 명령은 `endpoint-url` 파라미터를 사용해 VPC 엔드포인트를 지정합니다. 이러한 명령을 사용하려면 VPC 엔드포인트 ID 예제를 본인 계정의 ID로 바꿉니다.

```
$ aws payment-cryptography list-keys --endpoint-url https://
vpce-1234abcdf5678c90a-09p7654s-us-east-1a.ec2.us-east-1.vpce.amazonaws.com
```

VPC 엔드포인트를 만들 때 프라이빗 호스트 이름을 사용하도록 설정한 경우 CLI 명령 또는 애플리케이션 구성에 VPC 엔드포인트 URL을 지정할 필요가 없습니다. 표준 AWS Payment Cryptography DNS 호스트 이름은 VPC 엔드포인트로 확인됩니다. AWS CLI 및 SDKs는 기본적으로이 호스트 이름을 사용하므로 스크립트 및 애플리케이션에서 아무것도 변경하지 않고 VPC 엔드포인트를 사용하여 AWS Payment Cryptography 리전 엔드포인트에 연결할 수 있습니다.

프라이빗 호스트 이름을 사용하려면 VPC의 `enableDnsHostnames` 및 `enableDnsSupport` 속성을 `true`로 설정해야 합니다. 이러한 속성을 설정하려면 ModifyVpcAttribute 작업을 사용합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC에 대한 DNS 속성 보기 및 업데이트 섹션을 참조하세요.

## VPC 엔드포인트에 대한 액세스 제어

AWS Payment Cryptography용 VPC 엔드포인트에 대한 액세스를 제어하려면 VPC 엔드포인트 정책을 VPC 엔드포인트에 연결합니다. 엔드포인트 정책은 보안 주체가 VPC 엔드포인트를 사용하여 특정 AWS Payment Cryptography 리소스로 AWS Payment Cryptography 작업을 호출할 수 있는지 여부를 결정합니다.

엔드포인트를 생성할 때 VPC 엔드포인트 정책을 생성할 수 있으며, 언제든지 VPC 엔드포인트 정책을 변경할 수 있습니다. VPC 관리 콘솔이나 CreateVpcEndpoint 또는 ModifyVpcEndpoint 작업을 사용합니다. AWS CloudFormation 템플릿을 사용하여 VPC 엔드포인트 정책을 생성하고 변경할 수도 있습니다. VPC 관리 콘솔 사용에 대한 도움말은 AWS PrivateLink 설명서의 인터페이스 엔드포인트 생성 및 인터페이스 엔드포인트 수정을 참조하세요.

JSON 정책 문서 작성 및 형식 지정에 대한 도움말은 IAM 사용 설명서의 IAM JSON 정책 참조를 참조하세요.

주제

- VPC 엔드포인트 정책 정보
- 기본 VPC 엔드포인트 정책
- VPC 엔드포인트 정책 생성
- VPC 엔드포인트 정책 보기

## VPC 엔드포인트 정책 정보

VPC 엔드포인트를 사용하는 AWS Payment Cryptography 요청이 성공하려면 보안 주체에 다음 두 소스의 권한이 필요합니다.

- 자격 증명 기반 정책은 보안 주체에게 리소스(AWS 결제 암호화 키 또는 별칭)에 대한 작업을 호출할 수 있는 권한을 부여해야 합니다.
- VPC 엔드포인트 정책은 보안 주체에 엔드포인트를 사용하여 요청을 수행할 권한을 부여해야 합니다.

예를 들어 키 정책은 보안 주체에게 특정 AWS Payment Cryptography 키에서 Decrypt를 호출할 수 있는 권한을 부여할 수 있습니다. 그러나 VPC 엔드포인트 정책은 해당 보안 주체가 엔드포인트`Decrypt`를 사용하여 해당 AWS Payment Cryptography 키를 호출하도록 허용하지 않을 수 있습니다.

또는 VPC 엔드포인트 정책은 보안 주체가 엔드포인트를 사용하여 특정 AWS Payment Cryptography 키에서 StopKeyUsage를 호출하도록 허용할 수 있습니다. 그러나 보안 주체에게 IAM 정책의 권한이 없는 경우 요청이 실패합니다.

## 기본 VPC 엔드포인트 정책

모든 VPC 엔드포인트에는 VPC 엔드포인트 정책이 있지만 정책을 지정할 필요는 없습니다. 정책을 지정하지 않으면 기본 엔드포인트 정책은 엔드포인트의 모든 리소스에 대한 모든 보안 주체의 모든 작업을 허용합니다.

그러나 AWS Payment Cryptography 리소스의 경우 보안 주체에게 IAM 정책에서 작업을 호출할 수 있는 권한도 있어야 합니다. 따라서 실제로 기본 정책에서는 보안 주체가 리소스에 대한 작업을 호출할 권한이 있는 경우 엔드포인트를 사용하여 해당 작업을 호출할 수도 있다고 말합니다.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

보안 주체가 허용된 작업의 하위 집합에 대해서만 VPC 엔드포인트를 사용할 수 있도록 허용하려면 VPC 엔드포인트 정책을 생성하거나 업데이트합니다.

## VPC 엔드포인트 정책 생성

VPC 엔드포인트 정책은 보안 주체가 VPC 엔드포인트를 사용하여 리소스에 대한 작업을 수행할 권한이 있는지 여부를 결정합니다. AWS Payment Cryptography 리소스의 경우 보안 주체에게 IAM 정책에서 작업을 수행할 수 있는 권한도 있어야 합니다.

각 VPC 엔드포인트 정책문에는 다음 요소가 필요합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

정책문은 VPC 엔드포인트를 지정하지 않습니다. 대신 정책이 연결되는 모든 VPC 엔드포인트에 적용됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

다음은 AWS Payment Cryptography에 대한 VPC 엔드포인트 정책의 예입니다. VPC 엔드포인트에 연결된 경우이 정책은 `ExampleUser`가 VPC 엔드포인트를 사용하여 지정된 AWS Payment Cryptography 키에서 지정된 작업을 호출하도록 허용합니다. 이와 같은 정책을 사용하기 전에 예제 보안 주체 및 키 식별자를 계정의 유효한 값으로 바꿉니다.

```
{
   "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 는 VPC 엔드포인트를 사용하는 모든 작업을 기록합니다. 그러나 CloudTrail 로그에는 다른 계정의 보안 주체가 요청한 작업 또는 다른 계정의 AWS Payment Cryptography 키에 대한 작업은 포함되지 않습니다.

따라서 외부 계정의 보안 주체가 VPC 엔드포인트를 사용하여 로컬 계정의 모든 키에 대한 AWS Payment Cryptography 작업을 호출하지 못하도록 하는 VPC 엔드포인트 정책을 생성할 수 있습니다.

다음 예제에서는 aws:PrincipalAccount 전역 조건 키를 사용하여 보안 주체가 로컬 계정에 있지 않은 한 모든 AWS Payment Cryptography 키의 모든 작업에 대한 모든 보안 주체에 대한 액세스를 거부합니다. 이와 같은 정책을 사용하기 전에 예제 계정 ID를 유효한 것으로 교체하세요.

```
{
  "Statement": [
    {
      "Sid": "AccessForASpecificAccount",
      "Principal": {"AWS": "*"},
      "Action": "payment-cryptography:*",
      "Effect": "Deny",
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalAccount": "111122223333"
        }
      }
    }
  ]
}
```

## VPC 엔드포인트 정책 보기

엔드포인트에 대한 VPC 엔드포인트 정책을 보려면 VPC 관리 콘솔 또는 DescribeVpcEndpoints 작업을 사용합니다.

다음 AWS CLI 명령은 지정된 VPC 엔드포인트 ID가 있는 엔드포인트에 대한 정책을 가져옵니다.

이 명령을 사용하기 앞서 예제 엔드포인트 ID를 계정의 유효한 ID로 바꿉니다.

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpce-1234abcdf5678c90a`].[PolicyDocument]'
--output text
```

# 정책 설명에 VPC 엔드포인트 사용

요청이 VPC에서 왔거나 VPC 엔드포인트를 사용하는 경우 AWS Payment Cryptography 리소스 및 작업에 대한 액세스를 제어할 수 있습니다. 이렇게 하려면 IAM 정책 하나를 사용합니다.

- `aws:sourceVpce` 조건 키를 사용해 VPC 엔드포인트를 기반으로 액세스 권한을 부여하거나 액세스를 제한합니다.
- `aws:sourceVpc` 조건 키를 사용해 프라이빗 엔드포인트를 호스팅하는 VPC를 기반으로 액세스를 부여하거나 제한합니다.

> **Note**
>
> Amazon VPC 엔드포인트에서 요청을 보낼 때는 `aws:sourceIP` 조건 키가 유효하지 않습니다. 요청을 VPC 엔드포인트로 제한하려면 `aws:sourceVpce` 또는 `aws:sourceVpc` 조건 키를 사용합니다. 자세한 내용은AWS PrivateLink 가이드의 VPC 엔드포인트 및 VPC 엔드포인트 서비스에 대한 ID 및 액세스 관리 섹션을 참조하세요.

이러한 전역 조건 키를 사용하여 AWS Payment Cryptography 키, 별칭 및 특정 리소스에 의존하지 않는 CreateKey와 같은 작업에 대한 액세스를 제어할 수 있습니다.

예를 들어 다음 샘플 키 정책은 요청이 지정된 VPC 엔드포인트를 사용하는 경우에만 사용자가 AWS Payment Cryptography 키를 사용하여 특정 암호화 작업을 수행하도록 허용하여 인터넷 및 AWS PrivateLink 연결(설정된 경우)로부터의 액세스를 차단합니다. 사용자가 AWS Payment Cryptography에 요청하면 요청의 VPC 엔드포인트 ID가 정책의 `aws:sourceVpce` 조건 키 값과 비교됩니다. 두 값이 일치하지 않는 경우 요청이 거부됩니다.

이와 같은 정책을 사용하려면 자리 표시자 AWS 계정 ID 및 VPC 엔드포인트 IDs 계정의 유효한 값으로 바꿉니다.

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"
                }
            }
        }

    ]
}
```

aws:sourceVpc 조건 키를 사용하여 VPC 엔드포인트가 있는 VPC를 기반으로 AWS Payment Cryptography 키에 대한 액세스를 제한할 수도 있습니다.

다음 샘플 키 정책은에서 가져온 경우에만 AWS Payment Cryptography 키를 관리하는 명령을 허용합니다vpc-12345678. 또한에서 가져온 경우에만 암호화 작업에 AWS Payment Cryptography 키를 사용하는 명령을 허용합니다vpc-2b2b2b2b. 애플리케이션이 하나의 VPC에서 실행 중이지만 관리 용도로 VPC를 하나 더 사용하는 경우, 이와 같은 정책을 사용할 수 있습니다.

이와 같은 정책을 사용하려면 자리 표시자 AWS 계정 ID 및 VPC 엔드포인트 IDs 계정의 유효한 값으로 바꿉니다.

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": "*"
        }
    ]
}
```

## VPC 엔드포인트 로깅

AWS CloudTrail 는 VPC 엔드포인트를 사용하는 모든 작업을 기록합니다. AWS Payment Cryptography에 대한 요청이 VPC 엔드포인트를 사용하는 경우 요청을 기록하는 AWS CloudTrail 로그 항목에 VPC 엔드포인트 ID가 나타납니다. 엔드포인트 ID를 사용하여 AWS Payment Cryptography VPC 엔드포인트의 사용을 감사할 수 있습니다.

VPC를 보호하기 위해 VPC 엔드포인트 정책에 의해 거부되지만 그렇지 않으면 허용되었을 요청은에 기록되지 않습니다AWS CloudTrail.

예를 들어이 샘플 로그 항목은 VPC 엔드포인트를 사용한 GenerateMac 요청을 기록합니다. 로그 항목 끝에 `vpcEndpointId` 필드가 나타납니다.

```
{
      "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",
```

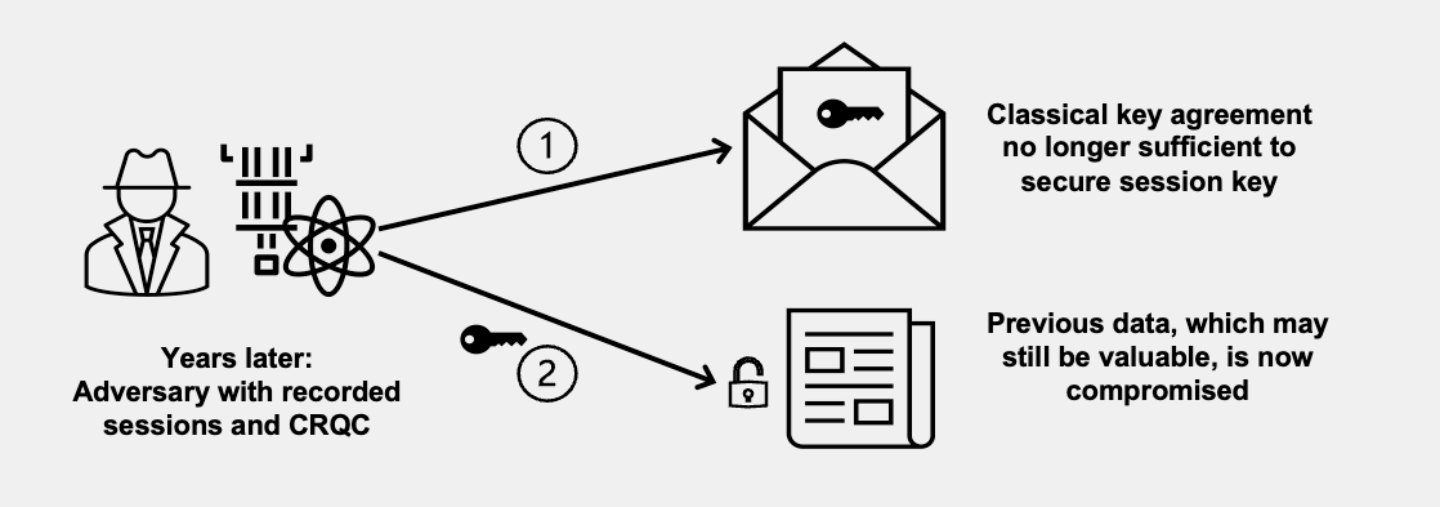
```
                "arn": "arn:aws:iam::111122223333:role/samplerole",
                "accountId": "111122223333",
                "userName": "samplerole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-05-27T19:34:10Z",
                "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"
    }
}
```

# 하이브리드 포스트 양자 TLS 사용

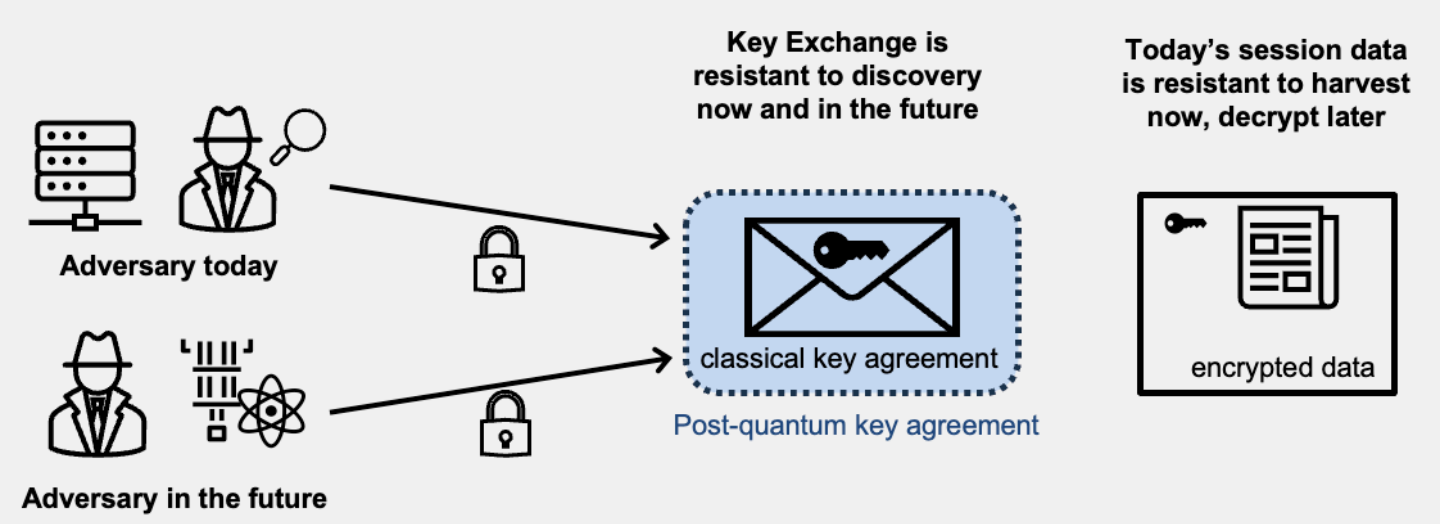
AWS Payment Cryptography 및 기타 많은 서비스는 전송 계층 보안(TLS) 네트워크 암호화 프로토콜에 대한 하이브리드 포스트 양자 키 교환 옵션을 지원합니다. API 엔드포인트에 연결하거나 AWS SDKs. 선택 사항인 이 하이브리드 포스트 양자 키 교환 기능은 적어도 우리가 현재 사용하는 TLS 암호화만큼 안전하며 장기적인 보안 이점을 추가로 제공할 수도 있습니다.

활성화된 서비스로 전송하는 데이터는 전송 계층 보안(TLS) 연결에서 제공하는 암호화로 전송 중에 보호됩니다. AWS Payment Cryptography가 TLS 세션에 지원하는 RSA 및 ECC를 기반으로 하는 클래식 암호 제품군은 키 교환 메커니즘에 대한 무차별 대입 공격을 현재 기술로 실행할 수 없게 만듭니다. 그러나 향후 대규모 또는 암호화적으로 관련된 양자 컴퓨터(CRQC)가 실용적이 되는 경우 기존 TLS 키 교환 메커니즘은 이러한 공격에 취약합니다. 공격자가 향후 복호화(지금 수집, 나중에 복호화)할 수 있기를 바라면서 지금 암호화된 데이터 수집을 시작할 수 있습니다. TLS 연결을 통해 전달되는 데이터의 장기 기밀성에 의존하는 애플리케이션을 개발하는 경우 대규모 양자 컴퓨터를 사용할 수 있게 되기 전에 양자 사후 암호화로 마이그레이션할 계획을 고려해야 합니다. AWS 는이 미래를 준비하기 위해 노력하고 있으며, 사용자도 잘 준비되기를 바랍니다.




현재 암호화된 데이터를 잠재적 향후 공격으로부터 보호하기 위해 AWS 는 양자 저항 또는 포스트 양자 알고리즘 개발에 암호화 커뮤니티에 참여하고 있습니다. AWS 는 클래식 및 포스트 양자 요소를 결합하는 하이브리드 포스트 양자 키 교환 암호 제품군을 구현하여 TLS 연결이 최소한 클래식 암호 제품군과 마찬가지로 강력하도록 합니다.

이러한 하이브리드 암호 제품군은 최신 버전의 AWS SDKs. 이 동작을 활성화/비활성화하는 방법에 대한 자세한 내용은 섹션을 참조하세요. ???




## TLS의 하이브리드 포스트 양자 키 교환 소개

AWS 가 사용하는 알고리즘은 오늘날 TLS에 사용되는 클래식 키 교환 알고리즘인 타원 곡선 디피-헬만(ECDH)과 미국 국립 표준 기술 연구소(NISTML-KEM)가 첫 번째 표준 포스트 양자 키 계약 알고리즘으로 지정한 퍼블릭 키 암호화 및 키 설정 알고리즘인 Module-Lattice-Based 키 캡슐화 메커니즘()을 결합하는 하이브리드입니다. 이 하이브리드는 각 알고리즘을 독립적으로 사용하여 키를 생성합니다. 그런 다음 두 키를 암호화 방식으로 결합합니다.

## PQC에 대해 자세히 알아보기

국립 표준 기술 연구소(NIST)의 포스트 양자 암호화 프로젝트에 대한 자세한 내용은 포스트 양자 암호화를 참조하세요.

NIST 포스트 양자 암호화 표준화에 대한 자세한 내용을 확인하려면 포스트 양자 암호화 표준화를 참조하세요.

## 하이브리드 포스트 양자 TLS 활성화

AWS SDKs 및 도구에는 언어 및 런타임에 따라 다른 암호화 기능과 구성이 있습니다. AWS SDK 또는 도구가 현재 PQ TLS 지원을 제공하는 세 가지 방법이 있습니다.

주제

- 기본적으로 PQ TLSSDKs
- 옵트인 PQ TLS 지원
- 시스템 OpenSSL을 사용하는 SDKs

- PQ TLS를 지원할 계획이 없는 AWS SDKs 및 도구

## 기본적으로 PQ TLSSDKs

> **Note**
>
> 6-Nov-2025일부터 MacOS 및 Windows용 AWS SDK 및 기본 CRT 라이브러리는 TLS용 시스템 라이브러리를 사용하므로 해당 플랫폼의 PQ TLS 기능은 일반적으로 시스템 수준 지원에 의해 결정됩니다.

AWS SDK for Go

Go용 AWS SDK는 표준 라이브러리에서 제공하는 Golang의 자체 TLS 구현을 사용합니다. Golang은 v1.24부터 PQ TLS를 지원 및 선호하므로 Go용 AWS SDK 사용자는 Golang을 v1.24로 업그레이드하기만 하면 PQ TLS를 활성화할 수 있습니다.

JavaScript용 AWS SDK(브라우저)

JavaScript용 AWS SDK(브라우저)는 브라우저의 TLS 스택을 사용하므로 브라우저 런타임이 이를 지원하고 선호하는 경우 SDK는 PQ TLS를 협상합니다. Firefox는 v132.0에서 PQ TLS에 대한 지원을 시작했습니다. Chrome은 v131에서 PQ TLS에 대한 지원을 발표했습니다. Edge는 데스크톱용 v120 및 Android용 140에서 옵트인 PQ TLS를 지원합니다.

Node.js용 AWS SDK

Node.js v22.20(LTS) 및 v24.9.0부터 Node.js는 OpenSSL 3.5를 정적으로 연결하고 번들링합니다. 즉, 해당 버전 및 후속 버전에 대해 기본적으로 PQ TLS가 활성화되고 선호됩니다.

Kotlin용 AWS SDK

Kotlin SDK는 v1.5.78부터 Linux의 PQ TLS를 지원하고 선호합니다. Kotlin용 AWS SDK의 CRT 기반 클라이언트는 MacOS 및 Windows의 TLS용 시스템 라이브러리에 의존하기 때문에 PQ TLS에 대한 지원은 이러한 기본 시스템 라이브러리에 따라 달라집니다.

Rust용 AWS SDK

Rust용 AWS SDK는 각 서비스 클라이언트에 대해 고유한 패키지(Rust 에코시스템에서 “상자”라고 함)를 배포합니다. 이들은 모두 통합 GitHub 리포지토리에서 관리되지만 각 서비스 클라이언트는 자체 버

전 및 릴리스 주기를 따릅니다. 통합 SDK는 8/29/25에 PQ TLS 기본 설정을 릴리스했으므로 해당 날짜 이후에 릴리스된 개별 서비스 클라이언트 버전은 기본적으로 PQ TLS를 지원하고 선호합니다.

관련 crates.io 버전 URL(예: AWS Payment Cryptography의 )로 이동하여 29-Aug-25일 이후에 게시된 첫 번째 버전을 찾아 특정 서비스 클라이언트에 대해 PQ TLS를 지원하는 최소 버전을 확인할 수 있습니다. 29-Aug-25일 이후에 게시된 모든 서비스 클라이언트 버전에는 기본적으로 PQ TLS가 활성화되고 선호됩니다.

## 옵트인 PQ TLS 지원

AWS SDK for C++

기본적으로 C++ SDK는 libcurl 및 WinHttp와 같은 플랫폼 네이티브 클라이언트를 사용합니다. Libcurl은 일반적으로 TLS용 시스템 OpenSSL을 사용하므로 시스템 OpenSSL이 ≥ v3.5인 경우에만 기본적으로 PQ TLS가 활성화됩니다. C++ SDK v1.11.673 이상에서이 기본값을 재정의하고 기본적으로 PQ TLS를 지원하고 활성화하는 AwsCrtHttpClient에 옵트인할 수 있습니다.

옵트인 PQ TLS 구축에 대한 참고 사항 이 스크립트를 사용하여 SDK의 CRT 종속성을 가져올 수 있습니다. 소스에서 SDK 빌드는 여기에 설명되어 https://github.com/aws/aws-sdk-cpp/tree/main?tab=readme-ov-file#building-from-source있지만 몇 가지 추가 CMake 플래그가 필요할 수 있습니다.

```
-DUSE_CRT_HTTP_CLIENT=ON \
-DUSE_TLS_V1_2=OFF \
-DUSE_TLS_V1_3=ON \
-DUSE_OPENSSL=OFF \
```

Java용 AWS SDK

v2부터 Java용 AWS SDK는 PQ TLS를 수행하도록 구성할 수 있는 AWS 공통 런타임(AWS CRT) HTTP 클라이언트를 제공합니다. v2.35.11부터 AwsCrtHttpClient는 사용되는 모든 곳에서 기본적으로 PQ TLS를 활성화하고 선호합니다.

## 시스템 OpenSSL을 사용하는 SDKs

여러 AWS SDKs 및 도구는 TLS용 시스템의 libcrypto/libssl 라이브러리에 의존합니다. 가장 자주 사용되는 시스템 라이브러리는 OpenSSL입니다. OpenSSL은 버전 3.5에서 PQ TLS 지원을 지원하므로 PQ TLS에 대해 이러한 SDKs 및 도구를 구성하는 가장 쉬운 방법은 OpenSSL 3.5 이상이 설치된 운영 체제 배포에서 사용하는 것입니다.

OpenSSL 3.5를 사용하여 Docker를 지원하는 모든 시스템에서 PQ TLS를 활성화하도록 Docker 컨테이너를 구성할 수도 있습니다. Python에 대해 이를 설정하는 예제는 Python의 포스트 양자 TLS를 참조하세요.

CLI

AWS CLI 설치 관리자를 통한 PQ TLS 지원이 곧 제공될 예정입니다. 즉시 활성화하려면 운영 체제에 따라 다르고 PQ TLS를 활성화할 수 있는 AWS CLI용 대체 설치 관리자를 사용할 수 있습니다.

MacOS의 경우 Homebrew를 통해 AWS CLI를 설치하고 Homebrew에서 제공하는 OpenSSL이 버전 3.5 이상으로 업그레이드되었는지 확인합니다. 이 작업은 “brew install openssl@3.6”을 사용하여 수행하고 “brew list | grep openssl”을 사용하여 검증할 수 있습니다.

Ubuntu 또는 Debian Linux의 경우: 사용 중인 Linux 배포에 OpenSSL 3.5 이상이 시스템 OpenSSL로 설치되어 있는지 확인합니다. 그런 다음 apt 또는 PyPI를 사용하여 AWS CLI를 설치합니다. 이러한 사전 조건을 사용하면 apt 또는 PyPI에서 제공하는 AWS CLI가 PQ-TLS를 협상하도록 구성됩니다. 설치를 검증하기 위한 step-by-step 지침은 github 리포지토리 및 관련 블로그 게시물을 참조하세요.

PHP용 AWS SDK

PHP용 AWS SDK는 시스템 libssl/libcrypto를 사용합니다. PQ TLS를 사용하려면 OpenSSL 3.5 이상이 설치된 운영 체제 배포에서이 SDK를 사용합니다.

AWS SDK for Python(Boto3)

Python용 AWS SDK(Boto3)는 시스템 libssl/libcrypto를 사용합니다. PQ TLS를 사용하려면 OpenSSL 3.5 이상이 설치된 운영 체제 배포에서이 SDK를 사용합니다.

Ruby 용 AWS SDK

Ruby용 AWS SDK는 시스템 libssl/libcrypto를 사용합니다. PQ TLS를 사용하려면 OpenSSL 3.5 이상이 설치된 운영 체제 배포에서이 SDK를 사용합니다.

AWS SDK for .NET

Linux에서 .NET용 AWS SDK는 시스템 libssl/libcrypto를 사용합니다. PQ TLS를 사용하려면 OpenSSL 3.5 이상이 설치된 운영 체제 배포에서이 SDK를 사용합니다. Windows 및 MacOS에서는 .NET 10 및 Windows 11부터 PQ TLS를 사용할 수 있습니다. MacOS에서는 여기에 설명된 대로 Apple의 Network.framework를 옵트인하여 TLS 1.3 지원(PQ TLS의 사전 조건)을 활성화할 수 있습니다. 최소 .NET 버전이 10이라고 가정하면 PQ TLS를 활성화해야 합니다.

### PQ TLS를 지원할 계획이 없는 AWS SDKs 및 도구

현재 다음 언어 SDKs 및 도구를 지원할 계획은 없습니다.

- SAP용 AWS SDK
- Swift용 AWS SDK
- Windows PowerShell용 AWS 도구

## AWS Payment Cryptography의 보안 모범 사례

AWS Payment Cryptography는 IAM 정책, 키 정책 및 IAM 정책을 구체화하기 위한 광범위한 정책 조건 키 세트, 키 블록과 관련된 PCI PIN 규칙의 기본 제공 적용 등 암호화 키의 보호를 강화하고 의도한 용도로 사용하도록 하기 위해 기본 제공되거나 선택적으로 구현할 수 있는 많은 보안 기능을 지원합니다.

> **Important**
>
> 제공된 일반적인 지침은 완전한 보안 솔루션을 나타내지는 않습니다. 모든 상황에 적합한 것은 아니기 때문에 이들이 규범적인 것은 아닙니다.

- 키 사용 및 사용 모드: AWS 결제 암호화는 ANSI X9 TR 31-2018 상호 운용 가능한 보안 키 교환 키 블록 사양에 설명된 대로 PCI PIN 보안 요구 사항 18-3에 따라 키 사용 및 사용 모드 제한을 따르고 적용합니다. 이렇게 하면 단일 키를 여러 용도로 사용할 수 있는 기능이 제한되고 키 메타데이터(예: 허용된 작업)가 키 구성 요소 자체에 암호화 방식으로 바인딩됩니다. AWS Payment Cryptography는 키 암호화 키(TR31_K0_KEY_ENCRYPTION_KEY)를 데이터 복호화에도 사용할 수 없도록 이러한 제한을 자동으로 적용합니다. 자세한 내용은 AWS Payment Cryptography 키의 키 속성 이해 단원을 참조하세요.
- 대칭 키 구성 요소의 공유 제한: 대칭 키 구성 요소(예: 핀 암호화 키 또는 키 암호화 키)를 최대 한 개체와 공유하세요. 더 많은 엔터티 또는 파트너에게 민감한 자료를 전송해야 하는 경우 추가 키를 생성합니다. AWS Payment Cryptography는 대칭 키 구성 요소 또는 비대칭 프라이빗 키 구성 요소를 일반에 노출하지 않습니다.
- 별칭이나 태그를 사용하여 키를 특정 사용 사례 또는 파트너와 연결: 별칭을 사용하여 키와 연결된 사용 사례를 쉽게 나타낼 수 있습니다 (예: BIN 12345과 연결된 카드 인증 키를 나타내는 alias/BIN_12345_CVK). 유연성을 높이려면 bin=12345, use_case=acquiring, country=us, partner=foo와 같은 태그를 만들어 보세요. 별칭과 태그를 사용하여 사용 사례 발행과 획득 간의 액세스 제어를 적용하는 등 액세스를 제한할 수도 있습니다.

- 최소 권한 액세스 실행: IAM을 사용하여 개별 사용자가 키를 생성하거나 암호화 작업을 실행하는 것을 금지하는 등 개인이 아닌 시스템에 대한 프로덕션 액세스를 제한할 수 있습니다. IAM을 사용하여 취득자의 핀 생성 또는 검증 기능을 제한하는 등 사용 사례에 적합하지 않을 수 있는 명령과 키 모두에 대한 액세스를 제한할 수도 있습니다. 최소 권한 액세스를 사용하는 또 다른 방법은 민감한 작업(예: 키 가져오기)을 특정 서비스 계정으로 제한하는 것입니다. AWS Payment Cryptography 자격 증명 기반 정책 예제의 예제를 참조하세요.

참고 항목

- AWS Payment Cryptography의 ID 및 액세스 관리
- IAM 사용 설명서의 IAM 보안 모범 사례

# AWS Payment Cryptography에 대한 규정 준수 검증

다른 AWS 서비스와 마찬가지로 고객은 보안 및 규정 준수를 위한 공동 책임 모델을 명확하게 이해해야 합니다. 특히 결제를 지원하는 서비스인 해당 PCI 표준 준수는 AWS Payment Cryptography 고객을 이해하는 데 특히 중요합니다. AWS PCI DSS 및 PCI 3DS 평가에는 AWS Payment Cryptography가 포함됩니다. 이러한 보고서에 대해 공동 책임 안내서의 서비스에 AWS Artifact대한 참조가 있을 수 있습니다. PCI PIN 보안 및 P2PE(Point-to-Point Encryption) 평가는 AWS Payment Cryptography에만 해당됩니다.

이 섹션에서는 서비스의 규정 준수 상태 및 범위에 대한 정보와 애플리케이션의 PCI PIN 보안 및 PCI P2PE 평가를 계획하는 데 도움이 되는 정보를 제공합니다.

주제

- 서비스 규정 준수
- PIN 규정 준수 계획
- P2PE 솔루션에서 AWS Payment Cryptography Decryption Component 사용

## 서비스 규정 준수

타사 감사자는 여러 규정 준수 프로그램의 일환으로 AWS Payment Cryptography의 보안 및 AWS 규정 준수를 평가합니다. 여기에는 SOC, PCI 등이 포함됩니다.

AWS Payment Cryptography는 PCI DSS 및 PCI 3DS 외에도 여러 PCI 표준에 대해 평가되었습니다. 여기에는 PCI PIN 보안(PCI PIN) 및 PCI 포인트 투 포인트(P2PE) 암호화가 포함됩니다. 사용 가능한 증명 및 규정 준수 가이드 AWS Artifact 는 섹션을 참조하세요.

특정 규정 준수 프로그램 범위의 AWS 서비스 목록은 규정 준수 프로그램 제공 범위 내 AWS 서비스규정 준수 프로그램 참조하세요. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하세요.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact을 참조하세요.

AWS Payment Cryptography 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다.는 규정 준수를 지원할 다음과 같은 리소스를 AWS 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서-이 배포 안내서에서는 아키텍처 고려 사항에 관해 설명하고 AWS에서 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 산업 및 위치에 적용될 수 있습니다.

- AWS Config 개발자 가이드의 규칙을 사용하여 리소스 평가-AWS Config, 리소스 구성이 내부 사례, 업계 지침, 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub CSPM-이 AWS 서비스는 보안 업계 표준 및 모범 사례 준수 여부를 확인하는 데 도움이 AWS 되는 내 보안 상태에 대한 포괄적인 보기를 제공합니다.

# PIN 규정 준수 계획

이 가이드에서는 AWS Payment Cryptography를 사용하는 PIN 처리 애플리케이션의 PCI PIN 평가를 준비해야 하는 문서와 증거를 설명합니다.

다른 AWS 서비스 및 규정 준수 표준과 마찬가지로 서비스를 안전하게 사용하고, 액세스 제어를 구성하고, PCI PIN 요구 사항에 따라 보안 파라미터를 사용하는 것은 사용자의 책임입니다. 이 가이드에서는 요구 사항을 충족하는 데 적합한 경우 이러한 구성에 대해 설명합니다.

주제

- 공통 주제
- 평가 범위
- 트랜잭션 처리 작업

## 공통 주제

애플리케이션을 HSM 연결에서 AWS Payment Cryptography와 같은 관리형 서비스로 마이그레이션하면 고객과 평가자에게 일반적인 문제와 개념이 발생합니다. 이 섹션에서는 서비스의 안전한 사용이 이러한 상황을 해결하는 방법을 명확히 하기 위한 정보를 제공합니다.

주제

- 공동 책임
- 최소 HSM 구성
- 고객과 APC 간의 키 교환

### 공동 책임

애플리케이션에 대한 완전한 보안 및 규정 준수 책임을 맡은 고객은 AWS Payment Cryptography의 키 관리, 보안 제어 및 관리형 HSM 기능(“서비스”)을 활용하도록 규정 준수를 재구성합니다. 이렇게 하면 AWS Payment Cryptography의 타사 평가에서 증명한 AWS대로 일부 요구 사항이 로 완전히 전환됩니

다. 일부 요구 사항은 고객의 애플리케이션과 서비스 간에 공유됩니다. 애플리케이션은 다음을 담당합니다.

- 서비스에 정확한 정보 제공
- 서비스의 권장 사항 및 PCI PIN 보안 요구 사항에 따라 보안 제어 사용
- 서비스에서 제공하는 도구를 사용하여 필요한 보안 제어 구현

고객과 평가자는에 규정 준수 증명과 함께 게시된 공동 책임 및 구현 가이드를 사용하여 제어 및 제어 모니터링을 AWS Artifact 구현한 다음 평가를 계획하고 완료합니다.

## 최소 HSM 구성

다른 PCI 표준의 기본 표준인 PCI 데이터 보안 표준에서는 함수에 필요한 최소 기능으로 모든 시스템을 구성해야 합니다. PCI PIN, P2PE 및 기타 솔루션 표준은 솔루션의 HSMs에이 요구 사항을 적용합니다. HSMs 솔루션에 필요한 함수만 활성화해야 합니다.

AWS 서비스는 시스템으로 취급하고 최소 필수 기능을 위해 구성해야 합니다. AWS의 Payment Card Industry Data Security Standard(PCI DSS) v4.0에서는 IAM을 사용하여 솔루션에서 사용하는 각 AWS 서비스에 대한 최소 기능을 구성하는 것이 좋습니다. 이는 AWS Payment Cryptography에도 적용됩니다. IAM 정책을 사용하면 암호화 함수를 사용하는 애플리케이션 구성 요소로만 암호화 함수를 제한할 수 있는 세분화된 권한을 사용할 수 있습니다.

## 고객과 APC 간의 키 교환

PIN PIN 보안 요구 사항 8-4 및 15-2에서는 키 교환 및 로드에 대한 퍼블릭 키가 인증되고 무결성이 보호되어야 합니다. ANSI/ASC X9 TR-34에서 기능적으로 설명되고 PCI PIN 부속서 A에서 관리하는 POI의 원격 키 로드의 경우, 퍼블릭 키는 부속서 A2-compliant 인증 기관에서 서명한 인증서로 전달되는 경우가 많습니다. 조직 간 교환의 경우 퍼블릭 키는 신뢰성과 무결성을 위한 다른 메커니즘을 사용합니다.

고객과 AWS 간의 모든 상호 작용은 각 APIs 상호 인증하고 TLS를 사용하여 호출 및 응답의 무결성을 보장하는 AWS API를 통해 이루어집니다. 고객 애플리케이션의 인증은 보안 토큰 및 SigV4와 같은 메커니즘을 사용하여 AWS Identity and Access Management에서 관리합니다. SigV4 AWS API 엔드포인트는 AWS SDKs. 그런 다음 TLS는 고객과 각 AWS API 간에 전달되는 모든 데이터의 기밀성과 무결성을 보장합니다.

APC APIs GetParametersForImport 및 ImportKey는 고객에서 서비스로의 키 전송을 구현합니다. GetParametersForImport에서 제공하는 인증 기관(CA)은 부속서 A2-compliant 않지만 안전하고 계정에 고유합니다. 이 CA는 요구 사항 8-4 및 15-2를 준수하기 위해에 의존할 수 없지만 가져온 키의 무결

성 확인을 제공합니다. GetCertificateSigningRequest API를 활용하여 자체 CA를 사용할 수도 있습니다.

퍼블릭 키 인증 및 무결성 보장을 제공하는 메커니즘은 다음과 같습니다.

- AWS API 인증에서 제공하는 인증
- 키의 무결성은 인증서의 자격 증명 정보가 신뢰할 수 없는 경우에도 GetParametersForImport에서 제공하는 인증서의 MAC 기능에 의해 제공됩니다. TLS가 고객과 AWS 간의 세션을 보호하는 데 사용하는 MAC에서도 키의 무결성을 보장합니다.

APC에서 제공하는 인증서 및 키 블록은 비대칭 메서드에 의한 인증서 및 키 보호 요구 사항을 지정하는 부속서 A1을 준수합니다.

## 평가 범위

평가를 계획하는 첫 번째 단계는 범위를 문서화하는 것입니다. PCI PIN의 경우 범위는 PINs을 보호하는 시스템 및 프로세스입니다. 여기에는 PIN을 보호하는 암호화 키 및 디바이스, 즉 POI(Pointpoints-of-interaction), HSMs 및 기타 보안 암호화 디바이스(SCD)라고도 하는 결제 터미널의 보호가 포함됩니다.

이러한 영역은 서비스 범위를 벗어나는 영역을 다루기 때문에 사용자가 전적인 책임을 지는 요구 사항은 다루지 않습니다. 결제 터미널의 구성 및 프로비저닝을 예로 들 수 있습니다. 에서 제공되는 PCI PIN에 대한 AWS Payment Cryptography 공동 책임 가이드를 참조하세요. AWS Artifact

주제

- 공동 책임
- 상위 수준 네트워크 다이어그램
- 키 테이블
- 문서 참조

### 공동 책임

AWS Payment Cryptography는 Visa PIN 보안 프로그램에 의해 정의되고 Visa Global Service Provider Registry의 “Amazon Web Services, LLC”에 등록된 암호화 및 지원 조직(ESO) 및 PIN 획득 타사 서비스 제공업체(TPS)입니다. 즉, Visa는 고객 PIN 평가자(PCI 적격 PIN 평가자 또는 PCI QPA)의 추가 평가 없이 PIN 획득 타사 VisaNet 프로세서(VNP), 서비스 공급자 역할을 하는 PIN 획득 클라이언트 VisaNet 프로세서, 기타 TPS 및 ESO 공급자가 서비스를 사용할 수 있도록 허용합니다.

다른 카드 브랜드 또는 결제 네트워크 공급자는 Visa PIN 보안 프로그램에 의존하거나 자체 프로그램을 보유할 수 있습니다. 다른 결제 네트워크 프로그램의 서비스 규정 준수에 대한 질문은 AWS Support에 문의하세요.

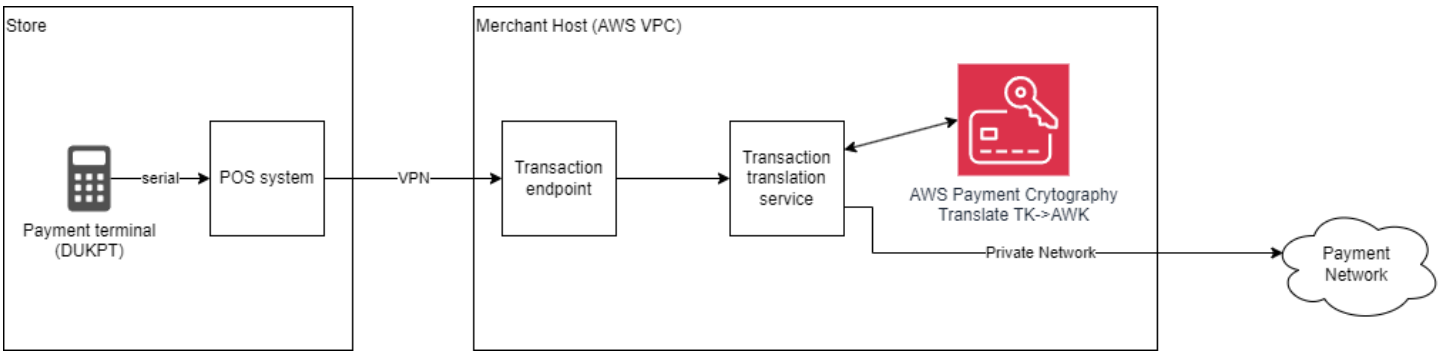
AWS 는 PCI PIN 보안 규정 준수 증명(AOC)과의 AWS Payment Cryptography에 대한 공동 책임 안내서를 제공합니다 AWS Artifact. PIN 처리에 서비스 공급자를 사용하는 것은 수년 동안 일반적이지만 PCI PIN 보안 표준은 버전 3.1까지 타사 서비스 공급자 관리를 다루지 않습니다. Visa PIN 보안 프로그램도 마찬가지입니다. 고객 QPA는 PCI DSS AOC 및 관련 요구 사항에 대한 테스트에 성공한 AWS것으로 규정 준수를 참조하는 공동 책임 가이드에 따라 수립된 모델을 따랐습니다.

## 상위 수준 네트워크 다이어그램

PCI PIN 보고 템플릿에는 “PIN 기반 트랜잭션 처리에 관여하는 엔터티의 경우 연결된 키 유형 사용과 함께 PIN 기반 트랜잭션 흐름을 설명하는 네트워크 도식이 제공됩니다. 또한 비대칭 기술을 사용하여 원격 키 배포에 참여하는 KIFs 및 엔터티는 키 구성 요소 흐름을 제공해야 합니다.”

AWS Payment Cryptography는 PCI PIN 평가를 위한 내부 서비스 구조를 보고했습니다. 다이어그램은 PIN 처리를 위해 서비스 APIs 보여줍니다.

AWS Payment Cryptography를 사용하는 PIN 애플리케이션에 대한 상위 수준 네트워크 다이어그램의 예:




## 키 테이블

보고서에는 PINs을 보호하는 모든 키가 직접 또는 간접적으로 나열되어야 합니다. 서비스에 있는 모든 키는 ListKeysAPI 에 나열할 수 있습니다.

애플리케이션의 키를 소유한 모든 리전 및 계정의 키 목록을 제공해야 합니다.

## 문서 참조

AWS Payment Cryptography의 안전한 사용을 위한 공급업체 설명서 및 권장 사항은 사용 설명서 및 API 참조에 나와 있습니다. 이는이 지침에서 적절하게 연결됩니다.

# 트랜잭션 처리 작업

PCI PIN 요구 사항은 제어 목표에 구성되어 있습니다. 각 Control Objective는 PINs.

주제

- 제어 목표 1: 이러한 요구 사항이 적용되는 트랜잭션에 사용되는 PINs은 보안을 유지하는 장비 및 방법을 사용하여 처리됩니다.
- 제어 목표 2: PIN 암호화/암호 해독 및 관련 키 관리에 사용되는 암호화 키는 키를 예측할 수 없거나 특정 키가 다른 키보다 더 가능성이 높은지 확인하는 프로세스를 사용하여 생성됩니다.
- 제어 목표 3: 키는 안전한 방식으로 전달되거나 전송됩니다.
- 제어 목표 4: HSMs 및 POI PIN 수락 디바이스에 대한 키 로드는 안전한 방식으로 처리됩니다.
- 제어 목표 5: 키는 무단 사용을 방지하거나 감지하는 방식으로 사용됩니다.
- 제어 목표 6: 키는 안전한 방식으로 관리됩니다.
- 제어 목표 7: PINs 및 키를 처리하는 데 사용되는 장비를 안전한 방식으로 관리합니다.

## 제어 목표 1: 이러한 요구 사항이 적용되는 트랜잭션에 사용되는 PINs은 보안을 유지하는 장비 및 방법을 사용하여 처리됩니다.

요구 사항 1: AWS Payment Cryptography에서 사용하는 HSMs PCI PIN 평가의 일부로 평가되었습니다. 서비스를 사용하는 고객의 경우 요구 사항 1-3 및 1-4는 서비스에서 관리하는 HSM을 기준으로 '실행 중'입니다. HSM에 대한 조사 결과에는 AWS QPA에서 테스트를 증명했음을 명시합니다. 규정 준수 PIN 증명을 참조할 수 있습니다 AWS Artifact. POI와 같은 솔루션의 다른 SCD는 인벤토리를 작성하고 참조해야 합니다.

요구 사항 2: 절차 문서에는 직원에 대한 공개, PINs 번역 프로토콜(들) 구현, 온라인 및 오프라인 처리 중 보호와 관련하여 카드 소지자 PIN이 보호되는 방법이 명시되어 있어야 합니다. 또한 설명서에는 각 영역 내에서 사용되는 암호화 키 관리 방법의 요약이 포함되어야 합니다.

요구 사항 3: 보안 PIN 암호화 및 전송을 위해 POI를 구성해야 합니다. AWS 결제 암호화는 요구 사항 3-3에 지정된 PIN 블록 변환만 지원합니다.

요구 사항 4: 애플리케이션이 PIN 블록을 저장해서는 안 됩니다. PIN 블록은 암호화된 경우에도 트랜잭션 저널 또는 로그에 보관해서는 안 됩니다. 서비스는 PIN 블록을 저장하지 않으며 PIN 평가는 해당 블록이 로그에 없는지 확인합니다.

PCI PIN 보안 표준은 표준에 명시된 대로 “ATM 및 point-of-sale) 터미널에서 온라인 및 오프라인 결제 카드 트랜잭션 처리 중에 개인 식별 번호(PIN) 데이터의 보안 관리, 처리 ATMs 및 전송”을 획득하는 데

적용됩니다. 그러나이 표준은 의도한 범위를 벗어나는 결제에 대한 암호화 키 관리를 평가하는 데 자주 사용됩니다. 여기에는 PINs 있습니다. 이러한 사례의 요구 사항에 대한 예외는 평가 대상과 합의해야 합니다.

### 제어 목표 2: PIN 암호화/암호 해독 및 관련 키 관리에 사용되는 암호화 키는 키를 예측할 수 없거나 특정 키가 다른 키보다 더 가능성이 높은지 확인하는 프로세스를 사용하여 생성됩니다.

요구 사항 5: AWS Payment Cryptography를 통한 키 생성은 PCI PIN 평가의 일부로 평가되었습니다. 이는 키 테이블 “생성자” 열에서 지정할 수 있습니다.

요구 사항 6: AWS Payment Cryptography에 보관된 키에 대한 보안 제어는 서비스의 PCI PIN 평가의 일부로 평가되었습니다. 애플리케이션 내의 키 생성 및 다른 서비스 공급자와 관련된 보안 제어에 대한 설명을 포함합니다.

요구 사항 7: 키 생성 방법을 지정하고 영향을 받는 모든 당사자가 이러한 절차/정책을 알고 있어야 하는 키 생성 정책 문서가 있어야 합니다. APC API를 사용한 키 생성 절차에는 키 생성 권한이 있는 역할 사용 및 키를 생성하는 스크립트 또는 기타 코드 실행에 대한 승인이 포함되어야 합니다. AWS CloudTrail 로그에는 날짜 및 시간, 키 ARN 및 사용자 ID가 있는 모든 CreateKey 이벤트가 포함됩니다. 물리적 미디어에 액세스하기 위한 HSM 일련 번호 및 로그는 서비스 PIN 평가의 일부로 평가되었습니다.

### 제어 목표 3: 키는 안전한 방식으로 전달되거나 전송됩니다.

요구 사항 8: AWS Payment Cryptography를 사용한 주요 전달은 PCI PIN 평가의 일부로 평가되었습니다. AWS Payment Cryptography에서 가져오기 전과 내보내기 후의 전송에 대한 키 보호 메커니즘을 문서화해야 합니다. 서비스는 모든 키에 대한 키 검사 값을 제공하여 올바른 전달을 검증합니다.

요구 사항 8-4에서는 퍼블릭 키를 무결성과 신뢰성을 보호하는 방식으로 전달해야 합니다. 애플리케이션과 간의 전달 AWS 은 AWS Identity and Access Management TLS 서버 인증서를 통해 메서드를 AWS사용하여 애플리케이션에 대한 AWS API 엔드포인트 인증에 대한 애플리케이션의 인증에 의해 제어됩니다. 또한 AWS Payment Cryptography에서 내보내거나 Payment Cryptography로 가져온 퍼블릭 키에는 고객별 임시 CAs가 서명한 인증서가 있습니다(GetPublicKeyCertificate, GetParametersForImport 및 GetParametersForExport 참조). 이러한 CAs는 PCI PIN 보안 부속서 A2를 준수하지 않으므로 인증의 유일한 방법으로 사용할 수 없습니다. 그러나 인증서는 여전히 인증을 제공하는 IAM을 통해 퍼블릭 키에 대한 무결성 보장을 제공합니다.

비대칭 방법을 사용하여 비즈니스 파트너와 퍼블릭 키를 교환할 때는 예를 들어 보안 파일 교환 웹 사이트를 사용하여 통신 채널을 통해 비즈니스 인증을 제공해야 합니다.

요구 사항 9: 서비스가 일반 텍스트 키 구성 요소를 사용하거나 직접 지원하지 않습니다.

요구 사항 10: 서비스는 전달을 위해 키를 보호하는 상대적 키 강도를 적용합니다. AWS Payment Cryptography에서 내보내기 전과 후에 키 가져오기, 내보내기 및 생성에 정확한 API 및 TR-31 파라미터를 사용하여 키 전달을 수행할 책임은 사용자에게 있습니다. 키 전달 메커니즘과 전달에 사용되는 암호화 키 목록을 설명하는 문서화된 절차가 있어야 합니다.

요구 사항 11: 절차 설명서에 키가 전달되는 방법을 지정해야 합니다. AWS Payment Cryptography API를 사용한 키 전달 절차에는 키 가져오기 및 내보내기 권한이 있는 역할 사용 및 key. AWS CloudTrail logs를 생성하는 스크립트 또는 기타 코드를 실행하기 위한 승인이 포함되어야 합니다. 로그에는 모든 ImportKey 및 ExportKey 이벤트가 포함됩니다.

## 제어 목표 4: HSMs 및 POI PIN 수락 디바이스에 대한 키 로드는 안전한 방식으로 처리됩니다.

요구 사항 12: 구성 요소 또는 공유에서 키를 로드하는 것은 사용자의 책임입니다. HSM 기본 키 관리는 서비스의 PIN 평가의 일부로 평가되었습니다. AWS Payment Cryptography는 개별 공유 또는 구성 요소에서 키를 로드하지 않습니다. 암호화 세부 정보 섹션을 참조하세요.

요구 사항 13 및 14: 서비스에서 가져오기 전과 내보내기 후의 전송에 대한 키 보호를 설명해야 합니다.

요구 사항 15: AWS Payment Cryptography는 서비스의 모든 키에 대한 키 확인 값과 퍼블릭 키에 대한 무결성 보장을 제공합니다. 애플리케이션은 이러한 검사를 사용하여 서비스로 가져오거나 서비스에서 내보낸 후 키를 검증할 책임이 있습니다. 검증 메커니즘이 마련되어 있는지 확인하기 위해 절차를 문서화해야 합니다.

요구 사항 15-2에서는 퍼블릭 키를 무결성과 신뢰성을 보호하는 방식으로 로드해야 합니다. ImportKey는 GetParametersForImport와 함께 제공된 서명 인증서의 검증을 제공합니다. 제공된 인증서가 자체 서명된 경우 보안 파일 교환과 같은 별도의 메커니즘에서 인증을 제공해야 합니다.

요구 사항 16: 절차에 대한 설명서는 키가 서비스에 로드되는 방법을 지정해야 합니다. API를 사용한 키 가져오기 절차에는 Key. AWS CloudTrail logs에 모든 ImportKey 이벤트가 포함된 스크립트 또는 기타 코드를 실행하기 위한 키 가져오기 권한 및 승인이 있는 역할 사용이 포함되어야 합니다. 설명서에 로깅 메커니즘을 포함해야 합니다. 서비스는 모든 키에 대한 키 검사 값을 제공하여 올바른 키 로드를 검증합니다.

## 제어 목표 5: 키는 무단 사용을 방지하거나 감지하는 방식으로 사용됩니다.

요구 사항 17: 서비스는 키 공유 관계를 추적할 수 있는 키에 대해 태그 및 별칭과 같은 메커니즘을 제공합니다. 또한 키가 공유될 때 알려진 키 값이나 기본 키 값이 사용되지 않음을 보여주기 위해 키 검사 값을 별도로 유지해야 합니다.

요구 사항 18: 서비스는 GetKey 및 ListKeys를 통해 키 무결성 검사를 제공하고 AWS CloudTrail,를 통해 무단 대체를 감지하거나 당사자 간 키 동기화를 모니터링하는 데 사용할 수 있는 키 관리 이벤트를 제공합니다. 서비스는 키 블록에만 키를 저장합니다. AWS Payment Cryptography에서 내보내기 전과 후에 키 저장 및 사용에 대한 책임은 사용자에게 있습니다.

PIN 기반 트랜잭션 또는 예상치 못한 키 관리 이벤트를 처리하는 동안 불일치가 발생하는 경우 즉각적인 조사를 위한 절차를 마련해야 합니다.

요구 사항 19: 서비스는 키 블록에서만 키를 사용하여 모든 작업에 KeyUsage, KeyModeOfUse 및 기타 키 속성을 적용합니다. 여기에는 프라이빗 키 작업에 대한 제한이 포함됩니다. 암호화 또는 디지털 서명 확인과 같은 단일 용도로 퍼블릭 키를 사용해야 하지만 둘 다 사용해서는 안 됩니다. 프로덕션 및 테스트/개발 시스템에는 별도의 계정을 사용해야 합니다.

요구 사항 20:이 요구 사항에 대한 책임은 사용자에게 있습니다.

## 제어 목표 6: 키는 안전한 방식으로 관리됩니다.

요구 사항 21: AWS Payment Cryptography에서 키 저장 및 사용은 서비스의 PCI PIN 평가의 일부로 평가되었습니다. 주요 구성 요소 관련 스토리지 요구 사항의 경우 21-2 및 21-3에 설명된 대로 저장해야 합니다. 서비스에서 내보내기 전과 후에 정책 설명서에서 키 보호 메커니즘을 설명해야 합니다.

요구 사항 22: AWS Payment Cryptography의 주요 손상 절차는 서비스의 PCI PIN 평가의 일부로 평가되었습니다. 모니터링 및 알림에 대한 응답을 포함하여 주요 침해 탐지 및 대응 AWS 절차를 설명해야 합니다.

요구 사항 23: AWS Payment Cryptography는 변형 또는 기타 되돌릴 수 있는 키 계산 방법을 지원하지 않습니다. 고객이 사용할 수 있는 키는 절대 없습니다. 되돌릴 수 있는 키 계산 사용은 서비스의 PCI PIN 평가의 일부로 평가되었습니다.

요구 사항 24: 내부 보안 암호 및 프라이빗 키에 대한 폐기 관행 AWS Payment Cryptography는 서비스의 PCI PIN 평가의 일부로 평가되었습니다. APC에서 가져오기 전과 내보내기 후 키의 키 폐기 절차를 설명해야 합니다. 주요 구성 요소 관련 폐기 요구 사항(24-2.2 및 24-2.3)은 여전히 사용자의 책임입니다.

요구 사항 25: AWS Payment Cryptography 내의 보안 암호 및 프라이빗 키에 대한 액세스는 서비스의 PCI PIN 평가의 일부로 평가되었습니다. AWS Payment Cryptography에서 내보내기 전과 후에 키에 대한 액세스 제어를 위한 프로세스와 설명서가 있어야 합니다.

요구 사항 26: 서비스 외부에서 사용되는 키, 키 구성 요소 또는 관련 자료에 대한 모든 액세스에 대한 로깅을 설명해야 합니다. 애플리케이션이 서비스로 수행하는 모든 키 관리 활동에 대한 로그는를 통해 제공됩니다 AWS CloudTrail.

요구 사항 27: 서비스 외부에서 사용되는 키, 키 구성 요소 또는 관련 자료에 대한 백업 절차를 설명해야 합니다.

요구 사항 28: API를 사용한 모든 키 관리 절차에는 키 관리 권한이 있는 역할 사용 및 키를 관리하는 스크립트 또는 기타 코드 실행에 대한 승인이 포함되어야 합니다. AWS CloudTrail 로그에는 모든 키 관리 이벤트가 포함되어야 합니다.

### 제어 목표 7: PINs 및 키를 처리하는 데 사용되는 장비를 안전한 방식으로 관리합니다.

요구 사항 29: AWS Payment Cryptography를 사용하여 HSMs에 대한 물리적 및 논리적 보호 요구 사항을 충족합니다.

요구 사항 30: 애플리케이션은 POI 디바이스 요구 사항에 대한 모든 물리적 및 논리적 보호에 대한 책임을 집니다.

요구 사항 31: AWS Payment Cryptography에서 사용하는 보안 암호화 디바이스(SCD) 보호는 서비스의 PCI PIN 평가의 일부로 평가되었습니다. 애플리케이션에서 사용하는 다른 SCDs에 대한 보호를 입증해야 합니다.

요구 사항 32: AWS Payment Cryptography에서 사용하는 SCDs 사용은 서비스의 PCI PIN 평가의 일부로 평가되었습니다. 애플리케이션에서 사용하는 다른 SCDs에 대한 액세스 제어 및 보호를 시연해야 합니다.

요구 사항 33: 사용자가 제어하는 모든 PIN 처리 장비의 보호를 설명해야 합니다.

## P2PE 솔루션에서 AWS Payment Cryptography Decryption Component 사용

PCI P2PE 솔루션은 AWS Payment Cryptography Decryption Component를 사용할 수 있습니다. 이 내용은 PCI Point-to-Point Encryption: Security Requirements and Testing Procedures, Section P2PE Solutions and Use of Third parties and/or P2PE Component Providers: “솔루션 공급자(또는 솔루션 공급자로서 판매자)는 PCI 목록에 있는 P2PE 구성 요소 공급자에 특정 P2PE 함수를 아웃소싱하고 P2PE Report on Validation(P-ROV)에서 PCI 목록에 있는 P2PE 구성 요소(들)의 사용을 보고할 수 있습니다.”에 설명되어 있습니다https://www.pcisecuritystandards.org/.

다른 AWS 서비스 및 규정 준수 표준과 마찬가지로 서비스를 안전하게 사용하고, 액세스 제어를 구성하고, PCI P2PE 요구 사항에 따라 보안 파라미터를 사용하는 것은 사용자의 책임입니다. 에서 사용할 수 있는 AWS Payment Cryptography P2PE 복호화 구성 요소 사용 설명서에 AWS Artifact는 AWS

Payment Cryptography를 PCI P2PE 솔루션 및 규정 준수 보고에 필요한 연간 복호화 구성 요소 보고서와 통합하기 위한 자세한 지침이 나와 있습니다.

# AWS Payment Cryptography의 ID 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 AWS Payment Cryptography 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제


- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- AWS Payment Cryptography가 IAM과 작동하는 방식
- AWS Payment Cryptography 자격 증명 기반 정책 예제
- AWS Payment Cryptography에 대한 리소스 기반 정책
- AWS Payment Cryptography에 대한 다자간 승인
- AWS Payment Cryptography 자격 증명 및 액세스 문제 해결


## 대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조AWS Payment Cryptography 자격 증명 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(AWS Payment Cryptography가 IAM과 작동하는 방식 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(AWS Payment Cryptography 자격 증명 기반 정책 예제 참조)

## ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증해야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

## AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자가 필요한 작업 목록은 IAM 사용자 설명서의 루트 사용자 자격 증명이 필요한 작업을 참조하세요.

## IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

## IAM 역할

IAM 역할은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

# 정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평

가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

## ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

## 리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

## 액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

## 기타 정책 타입

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 IAM 엔터티의 권한 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) – 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

## 여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

# AWS Payment Cryptography가 IAM과 작동하는 방식

IAM을 사용하여 AWS Payment Cryptography에 대한 액세스를 관리하기 전에 AWS Payment Cryptography에서 사용할 수 있는 IAM 기능을 이해해야 합니다. AWS Payment Cryptography 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

주제

- AWS Payment Cryptography 자격 증명 기반 정책
- AWS Payment Cryptography 태그 기반 권한 부여

## AWS Payment Cryptography 자격 증명 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부된 작업 및 리소스와 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. AWS Payment Cryptography는 특정 작업, 리소스 및 조건 키를 지원합

니다. JSON 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

### 작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

AWS Payment Cryptography의 정책 작업은 작업 앞에 접두사를 사용합니다`payment-cryptography:`. 예를 들어, AWS Payment Cryptography `VerifyCardData` API 작업으로 VPC를 생성할 수 있는 권한을 부여하려면 해당 정책에 `payment-cryptography:VerifyCardData` 작업을 포함합니다. 정책 문에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다. AWS Payment Cryptography는이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "payment-cryptography:action1",
      "payment-cryptography:action2"
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `List`라는 단어(`ListKeys` 및 `ListAliases` 등)로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "payment-cryptography:List*"
```

AWS Payment Cryptography 작업 목록을 보려면 IAM 사용 설명서의 AWS Payment Cryptography에서 정의한 작업을 참조하세요.

### 리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

결제 암호화 키 리소스에는 다음 ARN이 있습니다.

```
arn:${Partition}:payment-cryptography:${Region}:${Account}:key/${keyARN}
```

ARNs 형식에 대한 자세한 내용은 Amazon 리소스 이름(ARNs) 및 AWS 서비스 네임스페이스를 참조하세요.

예를 들어 문에서 `arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h` 인스턴스를 지정하려면 다음 ARN을 사용합니다.

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

특정 계정에 속하는 모든 키를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:payment-cryptography:us-east-2:111122223333:key/*"
```

키를 생성하기 위한 작업과 같은 일부 AWS Payment Cryptography 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

단일 명령문에서 여러 리소스를 지정하려면 아래와 같이 쉼표를 사용합니다.

```
"Resource": [
      "resource1",
      "resource2"
```

### 예제

AWS Payment Cryptography 자격 증명 기반 정책의 예를 보려면 섹션을 참조하세요AWS Payment Cryptography 자격 증명 기반 정책 예제.

## AWS Payment Cryptography 태그 기반 권한 부여

AWS Payment Cryptography 리소스에 태그를 연결하거나 AWS Payment Cryptography에 대한 요청에서 태그를 전달할 수 있습니다. 태그에 근거하여 액세스를 제어하려면 `payment-`

`cryptography:ResourceTag/key-name`, `aws:RequestTag/key-name`또는 `aws:TagKeys`조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

# AWS Payment Cryptography 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 AWS Payment Cryptography 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI또는 AWS API를 사용하여 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- AWS Payment Cryptography 콘솔 사용
- 사용자가 자신이 권한을 볼 수 있도록 허용
- AWS Payment Cryptography의 모든 측면에 액세스할 수 있는 기능
- 지정된 키를 사용하여 API 호출 가능
- 리소스를 구체적으로 거부할 수 있음

## 정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 AWS Payment Cryptography 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있

는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.

- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

## AWS Payment Cryptography 콘솔 사용

AWS Payment Cryptography 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은 AWS 계정의 AWS Payment Cryptography 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 최소 필수 권한보다 더 제한적인 보안 인증 기반 정책을 만들면 콘솔이 해당 정책에 연결된 개체(IAM 사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

이러한 엔터티가 AWS Payment Cryptography 콘솔을 계속 사용할 수 있도록 하려면 다음 AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

## 사용자가 자신이 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "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": "*"
        }
    ]
}
```

## AWS Payment Cryptography의 모든 측면에 액세스할 수 있는 기능

> **Warning**
>
> 이 예시는 광범위한 권한을 제공하므로 권장되지 않습니다. 대신 권한이 가장 낮은 액세스 모델을 고려해 보세요.

이 예제에서는 AWS 계정의 IAM 사용자에게 모든 AWS Payment Cryptography 키에 대한 액세스 권한과 `ControlPlane` 및 `DataPlane` 작업을 모두 포함한 모든 AWS Payment Cryptography API를 호출할 수 있는 권한을 부여하려고 합니다.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [
            {
               "Effect": "Allow",
               "Action": [
                  "payment-cryptography:*"
               ],
               "Resource": [
                  "*"
               ]
            }
      ]
   }
```

## 지정된 키를 사용하여 API 호출 가능

이 예제에서는 AWS 계정의 IAM 사용자에게 AWS Payment Cryptography 키 중 하나에 대한 액세스 권한을 부여한 다음 두 API`arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h`APIs`VerifyCardValidationData`. `GenerateCardValidationData` 반대로 IAM 사용자는 `DeleteKey` 또는 `ExportKey`와 같은 다른 작업에서 이 키를 사용할 수 없습니다.

리소스는 `key`가 앞에 붙은 키 또는 `alias`가 앞에 붙은 별칭일 수 있습니다.

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"
                ]
            }
        ]
    }
```

## 리소스를 구체적으로 거부할 수 있음

> **Warning**
>
> 와일드카드 액세스 권한 부여의 영향을 신중하게 고려하세요. 대신 최소 권한 모델을 고려해 보세요.

이 예제에서는 AWS 계정의 IAM 사용자가 AWS Payment Cryptography 키에 액세스할 수 있도록 허용하되 특정 키 하나에 대한 권한을 거부하려고 합니다. 사용자는 거부 문에 지정된 키를 제외한 모든 키로 `VerifyCardData` 및 `GenerateCardData`에 액세스할 수 있습니다.

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"
            ]
        }
    ]
}
```

# AWS Payment Cryptography에 대한 리소스 기반 정책

리소스 기반 정책은 AWS Payment Cryptography 키와 같은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책에서는 키에 액세스할 수 있는 사용자와 키에서 수행할 수 있는 작업을 지정합니다. 리소스 기반 정책을 사용하여 다음을 수행할 수 있습니다.

- 여러 사용자 및 역할에 단일 키에 대한 액세스 권한을 부여합니다.
- 다른 AWS 계정의 사용자 또는 역할에 액세스 권한을 부여합니다.

주제

- 고려 사항
- 리소스 기반 정책 관리
- 리소스 기반 정책 예제

리소스 기반 정책을 AWS Payment Cryptography 키에 연결하면 AWS Payment Cryptography는 IAM 정책 평가 로직을 사용하여 지정된 보안 주체가 요청된 작업을 수행할 권한이 있는지 확인합니다. 교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 엔터티를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 교차 계정 액세스에는 다음 두 가지 정책이 필요합니다.

1. 리소스 기반 정책(키 소유자의 계정) - 키 소유자는 `PutResourcePolicy`를 사용하여 호출자의 계정 또는 IAM 보안 주체에 대한 액세스 권한을 부여합니다.
2. 자격 증명 기반 정책(발신자의 계정) - 호출자의 IAM 관리자는 호출자의 IAM 정책에서 AWS Payment Cryptography 작업(예: `payment-cryptography:EncryptData`)도 허용해야 합니다.

두 정책 모두 작업을 허용해야 합니다. 둘 중 하나가 누락된 경우에서 교차 계정 요청이 거부됩니다AccessDeniedException.

리소스 기반 정책이 동일한 계정의 보안 주체에게 액세스 권한을 부여하는 경우 추가 자격 증명 기반 정책은 필요하지 않습니다. 자세한 내용은 IAM 사용자 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조합니다.

> ⚠ 리소스 정책 컨트롤 플레인 작업
>
> 리소스 기반 정책은 PutResourcePolicy, GetResourcePolicy및와 같은 리소스 정책 컨트롤 플레인 작업에는 적용되지 않습니다DeleteResourcePolicy. 이렇게 하면 리소스 정책이 정책 자체를 수정하거나 제거하는 기능을 거부할 수 있는 잠재적 잠금 시나리오를 방지할 수 있습니다. 이러한 컨트롤 플레인 작업에 대한 액세스는 IAM 자격 증명 기반 정책으로만 관리됩니다.

## 고려 사항

AWS Payment Cryptography에서 리소스 기반 정책을 사용할 때는 다음 사항에 유의하세요.

- AWS Payment Cryptography는 키에 대한 퍼블릭 액세스를 자동으로 적용하지 않습니다. 익명 또는 퍼블릭 보안 주체에게 액세스 권한을 부여하는 리소스 기반 정책은 생성할 수 없습니다. AWS Payment Cryptography 키에 대한 모든 액세스에는 인증된 보안 AWS 주체가 필요하며 퍼블릭 액세스는 항상 차단됩니다.
- 리소스 기반 정책은 키별로 적용됩니다. 각 AWS Payment Cryptography 키에는 최대 하나의 리소스 기반 정책이 연결될 수 있습니다.
- 리소스 기반 정책은 별칭에 적용되지 않습니다. 별칭으로 키를 참조하면 기본 키에 연결된 리소스 정책이 평가됩니다.
- 리소스 기반 정책은 현재 다중 리전 키 복제를 사용하여 생성된 읽기 전용 복제본 리전 키에는 적용되지 않습니다. 리소스 정책은 기본 리전 키에만 연결할 수 있습니다.
- 리소스 기반 정책의 Resource 요소는 정책이 연결된 키의 ARN"*"이거나 정확히 일치해야 합니다. 동일한 정책 문서를 여러 키에서 재사용할 수 있으므로를 사용하는 것이 "*" 좋습니다.
- 리소스 정책 관리 APIs(PutResourcePolicy, GetResourcePolicy및 DeleteResourcePolicy)는 키를 소유 AWS 계정 한 로 제한됩니다. 키 소유자 계정 내의 보안 주체만 리소스 정책을 관리할 수 있습니다.

# 리소스 기반 정책 관리

AWS CLI 또는 AWS API를 사용하여 AWS Payment Cryptography 키에 대한 리소스 기반 정책을 관리할 수 있습니다. 이 명령을 사용하려면 예제 명령의 *#### ## ### ####* 자신의 정보로 바꿉니다.

리소스 기반 정책 연결

PutResourcePolicy API 작업 또는 put-resource-policy CLI 명령을 사용하여 리소스 기반 정책을 키에 연결합니다. 정책이 이미 있는 경우 명령이 정책을 대체합니다.

다음 예제에서는 JSON 파일의 리소스 기반 정책을 키에 연결합니다.

```
aws payment-cryptography put-resource-policy \
    --resource-arn arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h \
    --policy file://policy.json
```

리소스 기반 정책 검색

GetResourcePolicy API 작업 또는 get-resource-policy CLI 명령을 사용하여 키에 연결된 리소스 기반 정책을 검색합니다.

다음 예시에서는 키에 연결된 리소스 기반 정책을 검색합니다.

```
aws payment-cryptography get-resource-policy \
    --resource-arn arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h
```

응답은 정책 문서를 반환합니다.

```
{
    "Policy": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
                },
                "Action": [
                    "payment-cryptography:EncryptData",
```

```
                "payment-cryptography:DecryptData"
            ],
            "Resource": "*"
        }
    ]
  }
}
```

리소스 기반 정책 삭제

DeleteResourcePolicy API 작업 또는 delete-resource-policy CLI 명령을 사용하여 키에서 리소스 기반 정책을 제거합니다.

다음 예시에서는 키에 연결된 리소스 기반 정책을 삭제합니다.

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

## 리소스 기반 정책 예제

### 키에 대한 교차 계정 액세스 권한 부여

다음 리소스 기반 정책은 다른 AWS 계정의 역할에 암호화 작업에 AWS Payment Cryptography 키를 사용할 수 있는 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
            },
            "Action": [
                "payment-cryptography:GenerateCardValidationData",
                "payment-cryptography:VerifyCardValidationData"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

### 다른 계정에 다른 권한 부여

다음 리소스 기반 정책은 별도의 계정의 보안 주체에게 다른 권한을 부여하는 방법을 보여줍니다. 이 예제에서는 한 계정의 3DS Access Control Server(ACS)가 카드 검증 데이터를 생성할 수 있는 반면, 다른 계정의 Payment Authorization 서비스는 3DS 암호화만 검증할 수 있습니다.

```
{
    "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": "*"
        }
    ]
}
```

## AWS Payment Cryptography에 대한 다자간 승인

AWS Payment Cryptography는 Amazon Web Services Organizations의 기능인 다자간 승인(MPA)과 통합되어 분산 승인 프로세스를 통해 중요한 작업을 보호합니다. MPA를 사용하면 신뢰할 수 있는 여러 개인이 특정 AWS Payment Cryptography 작업을 수행하기 전에 승인하도록 요구할 수 있습니다.

주제


- 개요
- 보호된 작업
- 사전 조건
- MPA 활성화 및 비활성화
- 시작하기
- 예: MPA가 활성화된 루트 인증서 가져오기
- AWS CloudTrail MPA 이벤트에 대한 로깅
- 요청 상태 확인 및 실패 처리


## 개요

다자간 승인은 작업을 진행하기 전에 신뢰할 수 있는 개인 그룹의 승인을 요구하여 민감한 AWS Payment Cryptography 작업에 보안 계층을 추가합니다. 이렇게 하면 단일 자격 증명 세트가 손상된 경우 무단 변경으로부터 보호하고 단일 개인이 일방적으로 변경하는 것을 방지할 수 있습니다.

승인 팀은 보호된 작업 요청을 승인하거나 거부하도록 지정한 조직 내 승인자 그룹입니다. 승인 프로세스는 전적으로 조직의 승인자가 관리합니다. 요청 승인 또는 거부에는 어떤 AWS 직원도 관여하지 않습니다.

보호된 작업에 대해 MPA가 활성화되면 다음이 발생합니다.

1. 요청자가 보호된 작업을 시작합니다.
2. MPA는 승인 세션을 생성하고 승인 팀원에게 알립니다.
3. 승인 팀원은 MPA 포털을 통해 요청을 검토하고 승인 또는 거부합니다.
4. 필요한 최소 승인 임계값에 도달하면 작업이 진행됩니다. 승인 팀이 요청을 거부하거나 승인 임계값에 도달하기 전에 허용된 세션 시간이 만료되면 작업이 수행되지 않습니다. 두 경우 모두 요청자는 작업을 재시도하기 위해 새 요청을 제출해야 합니다.

> **Note**
>
> MPA가 활성화된 루트 CA 인증서를 가져올 때 `RequesterComment` 파라미터는 필수입니다. 이 설명은 승인 팀에 전송된 승인 알림에 포함되어 요청에 대한 컨텍스트를 제공합니다.

## 보호된 작업

AWS Payment Cryptography는 다음 작업에 대해 MPA를 지원합니다.

- `ImportKey` `RootCertificatePublicKey` 키 구성 요소 사용 - 루트 퍼블릭 키 인증서를 가져오는 것은 중요한 작업입니다. 루트 인증서는 TR-34와 같은 비대칭 키 교환을 사용하여 이후의 모든 키 가져오기 및 내보내기에 대한 트러스트 앵커를 설정하기 때문입니다. 이 작업에 대한 다자간 승인을 요구하면 단일 개인이 AWS Payment Cryptography 키에 대한 신뢰 루트를 일방적으로 설정하거나 변경할 수 없도록 하는 데 도움이 됩니다.

## 사전 조건

AWS Payment Cryptography에서 MPA를 사용하려면 먼저 다음 사전 조건을 완료해야 합니다.

- Amazon Web Services Organizations 환경에서 MPA를 설정합니다. 지침은 다자간 승인이란 무엇입니까?를 참조하세요. 다자간 승인 사용 설명서의 .
- 필수 승인자가 있는 승인 팀을 하나 이상 생성합니다.
- 를 사용하여 AWS Payment Cryptography 키 AWS 계정 가 포함된와 승인 팀을 공유합니다 AWS Resource Access Manager.
- 조직의 관리 계정을 다자간 승인에 옵트인해야 합니다.

## MPA 활성화 및 비활성화

승인 팀을 설정한 후 팀을 계정과 연결하여 AWS Payment Cryptography용 MPA를 활성화할 수 있습니다. 연결을 해제하려면 현재 연결된 승인 팀의 승인이 필요하지만 팀을 연결 해제하여 MPA를 비활성화할 수도 있습니다.

MPA 활성화

`AssociateMpaTeam` API 작업 또는 associate-mpa-team CLI 명령을 사용하여 승인 팀을 AWS Payment Cryptography 계정과 연결합니다. 일단 연결되면 보호된 작업을 진행하기 전에 팀의 승인이 필요합니다.

```
aws payment-cryptography associate-mpa-team \
    --team-arn arn:aws:mpa:us-east-1:111122223333:team/my-approval-team
```

MPA 비활성화

DisassociateMpaTeam API 작업 또는 disassociate-mpa-team CLI 명령을 사용하여 승인 팀 연결을 제거합니다. 팀을 연결 해제하는 것은 그 자체로 현재 연결된 승인 팀의 승인이 필요한 보호 작업입니다.

```
aws payment-cryptography disassociate-mpa-team \
    --team-arn arn:aws:mpa:us-east-1:111122223333:team/my-approval-team
```

**Important**

MPA를 비활성화하려면 현재 연결된 승인 팀의 승인이 필요합니다. 이렇게 하면 단일 개인이 다자간 승인 보호를 일방적으로 제거할 수 없습니다.

**Note**

--requester-comment 파라미터는 associate-mpa-team 및의 경우 선택 사항입니다disassociate-mpa-team.

## 시작하기

AWS Payment Cryptography용 MPA를 시작하려면 승인 팀을 생성하고, 승인 정책을 구성하고, 승인 세션을 관리하는 방법을 포함한 자세한 설정 지침은 다자간 승인 사용 설명서를 참조하세요.

## 예: MPA가 활성화된 루트 인증서 가져오기

MPA가 활성화되고 승인 팀이에 대한 ImportKey 작업과 연결되면 RootCertificatePublicKey가져오기 요청을 진행하기 전에 승인이 필요합니다.

1. 요청자가를 호출import-key하여 루트 퍼블릭 키 인증서를 가져옵니다. 이 명령을 사용하려면 예제 명령의 #### ## ### #### 자신의 정보로 바꿉니다.

```
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"
```

응답은가 로 KeyState 설정된 키를 반환하여 요청이 승인을 기다리고 CREATE_IN_PROGRESS있음을 나타냅니다. 응답에는 승인 세션에 대한 세부 정보가 MpaStatus 포함됩니다.

```
{
    "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. MPA가 활성화되어 있으므로 요청이 즉시 완료되지 않습니다. 대신 AWS Payment Cryptography는 승인 세션을 생성하고 승인이 보류 중임을 나타내는 응답을 반환합니다.
3. 승인 팀원은 알림을 받고 MPA 포털을 통해 요청을 검토합니다. 필요한 수의 승인자가 요청을 승인하면 가져오기 작업이 진행되고 루트 인증서를 가져옵니다.

## AWS CloudTrail MPA 이벤트에 대한 로깅

MPA가 활성화되면 승인 세션이 완료될 AWS CloudTrail 때 AWS Payment Cryptography가 서비스 이벤트를에 기록합니다. 이러한 이벤트는 요청의 승인 또는 실패 여부를 포함하여 승인 프로세스의 결과

를 기록합니다. 이러한 로그를 사용하여 MPA 활동을 감사하고 보호된 작업의 상태를 추적할 수 있습니다.

MPA 관련 CloudTrail 이벤트에는의 `serviceEventDetails`다음 필드가 포함됩니다.

- `keyArn` - 작업의 영향을 받는 키의 ARN입니다.
- `operation` - 요청된 보호된 작업입니다.
- `mpaSessionArn` - MPA 승인 세션의 ARN입니다.
- `sessionStatus` - 승인 세션의 결과입니다(`APPROVED` 또는 `FAILED`).

### 승인된 요청

다음 예제는 MPA 팀이 승인한 `ImportKey` 요청에 대한 CloudTrail 이벤트를 보여줍니다.


```
{
    "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"
    }
}
```

### 실패한 요청

다음 예제는 거부되거나 시간 초과된 ImportKey 요청에 대한 CloudTrail 이벤트를 보여줍니다.

```
{
    "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"
    }
}
```

에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 AWS CloudTrail참조하세요.

## 요청 상태 확인 및 실패 처리

를 호출하여 보류 중인 MPA 요청의 상태를 확인할 수 있습니다GetKey. 응답에는 현재 승인 세션 세부 정보가 포함된 MpaStatus 필드가 포함됩니다. 이 명령을 사용하려면 예제 명령의 *#### ## ### ## ##* 자신의 정보로 바꿉니다.

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

요청이 승인 보류 중인 동안 응답은 KeyState CREATE_IN_PROGRESS 및 MpaStatus.Status로 표시됩니다PENDING.

```
{
    "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"
        }
    }
}
```

필요한 수의 승인자가 요청을 승인하면가 로 KeyState 이동CREATE_COMPLETE하고 로 MpaStatus.Status 이동합니다APPROVED. 이제 키를 사용할 준비가 되었습니다.

```
{
```

```
    "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"
        }
    }
}
```

승인 팀이 요청을 거부하거나 승인 임계값이 충족되기 전에 세션이 만료되면가 로 KeyState 변경CREATE_FAILED되고가 로 MpaStatus.Status 변경됩니다FAILED.

```
{
    "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"
```

```
        }
    }
}
```

CREATE_FAILED 상태의 키는 암호화 작업에 사용할 수 없습니다. 가져오기를 다시 시도하려면 새 ImportKey 요청을 제출해야 합니다. 그러면 새 승인 세션이 생성됩니다.

# AWS Payment Cryptography 자격 증명 및 액세스 문제 해결

AWS Payment Cryptography와 관련된 IAM 관련 문제가 확인되는 대로 이 섹션에 주제가 추가될 예정입니다. IAM 주제에 대한 일반적인 문제 해결 내용은 IAM 사용 설명서의 문제 해결 섹션을 참조하세요.

# AWS Payment Cryptography 모니터링

모니터링은 AWS Payment Cryptography 및 기타 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS는 AWS Payment Cryptography를 관찰하고, 문제가 있을 때 보고하고, 적절한 경우 자동 조치를 취할 수 있는 다음과 같은 모니터링 도구를 제공합니다.

- Amazon CloudWatch는 AWS 리소스와 실행 중인 애플리케이션을 AWS 실시간으로 모니터링합니다. 지표를 수집 및 추적하고, 사용자 지정 대시보드를 생성할 수 있으며, 지정된 지표가 지정한 임곗값에 도달하면 사용자에게 알리거나 조치를 취하도록 경보를 설정할 수 있습니다. 예를 들어 CloudWatch가 특정 APIs의 사용량을 추적하도록 하거나 AWS Payment Cryptography 할당량에 근접하는 경우 알림을 받을 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서를 참조하세요.
- Amazon CloudWatch Logs로 Amazon EC2 인스턴스, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임곗값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하세요.
- AWS CloudTrail는 AWS 계정에서 또는 계정을 대신하여 수행한 API 호출 및 관련 이벤트를 캡처하고 사용자가 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 호출된 사용자 및 계정 AWS, 호출된 엔드포인트, 사용된 리소스(키), 호출이 수행된 소스 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

주제

- 를 사용하여 AWS Payment Cryptography API 호출 로깅 AWS CloudTrail

## 를 사용하여 AWS Payment Cryptography API 호출 로깅 AWS CloudTrail

AWS Payment Cryptography는 AWS Payment Cryptography에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 AWS Payment Cryptography에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처되는 직접 호출에는 콘솔로부터의 직접 호출과 API 작업에 대한 코드 직접 호출이 포함됩니다. 추적을 생성하면 AWS Payment Cryptography 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 관리(컨트롤 플레인) 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 AWS Payment Cryptography

에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

주제

- AWS CloudTrail의 Payment Cryptography 정보
- CloudTrail의 컨트롤 플레인 이벤트
- CloudTrail의 데이터 이벤트
- AWS Payment Cryptography Control Plane 로그 파일 항목 이해
- AWS Payment Cryptography 데이터 영역 로그 파일 항목 이해

## AWS CloudTrail의 Payment Cryptography 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. AWS Payment Cryptography에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

AWS Payment Cryptography 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS Region에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 추가적으로, CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 수신
- 여러 계정에서 CloudTrail 로그 파일 수신

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

## CloudTrail의 컨트롤 플레인 이벤트

CloudTrail은 CreateKey, ImportKey, DeleteKey, ListKeys, TagResource 및 기타 모든 컨트롤 플레인 작업과 같은 AWS Payment Cryptography 작업을 기록합니다.

## CloudTrail의 데이터 이벤트

데이터 이벤트는 페이로드 암호화 또는 핀 변환과 같이 리소스에서 또는 리소스에서 수행되는 리소스 작업에 대한 정보를 제공합니다. 데이터 이벤트는 CloudTrail이 기본적으로 로깅하지 않는 대량 활동입니다. CloudTrail API 또는 콘솔을 사용하여 AWS Payment Cryptography 데이터 영역 이벤트에 대한 데이터 이벤트 APIs 작업 로깅을 활성화할 수 있습니다. 자세한 내용을 알아보려면 AWS CloudTrail 사용 설명서의 데이터 이벤트 로깅을 참조하세요.

CloudTrail에서는 고급 이벤트 선택기를 사용하여 로깅되고 기록되는 AWS Payment Cryptography API 활동을 결정해야 합니다. AWS Payment Cryptography 데이터 영역 이벤트를 로깅하려면 리소스 유형 `AWS Payment Cryptography key` 및를 포함해야 합니다`AWS Payment Cryptography alias`. 이렇게 설정하면 `eventName` 필터를 사용하여 `EncryptData` 이벤트를 추적하는 등 기록할 특정 데이터 이벤트를 선택하여 로깅 기본 설정을 더 세분화할 수 있습니다. 자세한 내용을 알아보려면 AWS CloudTrail API 참조의 *`AdvancedEventSelector`* 섹션을 참조하세요.

> **Note**
>
> AWS Payment Cryptography 데이터 이벤트를 구독하려면 고급 이벤트 선택기를 사용해야 합니다. 모든 이벤트를 수신하려면 키 및 별칭 이벤트를 구독하는 것이 좋습니다.

AWS Payment Cryptography 데이터 이벤트:

- `DecryptData`
- `EncryptData`
- `GenerateCardValidationData`

- GenerateMac
- GeneratePinData
- ReEncryptData
- TranslatePinData
- VerifyAuthRequestCryptogram
- VerifyCardValidationData
- VerifyMac
- VerifyPinData

데이터 이벤트에는 추가 요금이 적용됩니다. 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

## AWS Payment Cryptography Control Plane 로그 파일 항목 이해

트레일이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제에서는 AWS Payment Cryptography CreateKey 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
      {
    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={
    }
  }
}
```

다음 예제에서는 다중 리전 키 복제를 활성화하는 AWS Payment Cryptography를 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
    "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"
}
```

## AWS Payment Cryptography 데이터 영역 로그 파일 항목 이해

데이터 영역 이벤트는 선택적으로 구성하고 컨트롤 플레인 로그와 유사하게 작동할 수 있지만 일반적으로 훨씬 더 높은 볼륨입니다. AWS Payment Cryptography 데이터 영역 작업에 대한 일부 입력 및 출력의 민감한 특성을 고려할 때 "*** Sensitive Data Redacted ***" 메시지가 있는 특정 필드를 찾을 수 있

습니다. 이는 구성이 불가능하며 민감한 데이터가 로그 또는 추적에 나타나지 않도록 하기 위한 것입니다.

다음 예제에서는 AWS Payment Cryptography `EncryptData` 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

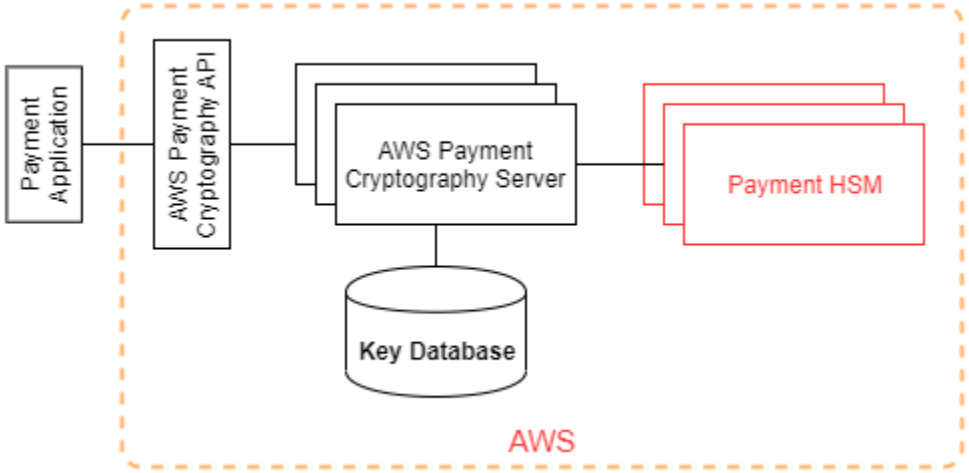
```
      {
        "Records": [
            {
                "eventVersion": "1.09",
                "userIdentity": {
                    "type": "AssumedRole",
                    "principalId": "TESTXECZ5U2ZULLHHMJG:DataPlane-User",
                    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/DataPlane-
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"
                }
            }
        ]
    }
```

# 암호화 세부 정보

AWS Payment Cryptography는 결제 트랜잭션을 위한 암호화 키를 생성하고 관리하는 웹 인터페이스를 제공합니다. AWS Payment Cryptography는 표준 키 관리 서비스와 결제 트랜잭션 암호화 및 중앙 집중식 관리 및 감사에 사용할 수 있는 도구를 제공합니다. 이 설명서에서는 서비스가 제공하는 기능을 평가하는 데 도움이 되도록 AWS Payment Cryptography에서 사용할 수 있는 암호화 작업에 대한 자세한 설명을 제공합니다.

AWS Payment Cryptography에는 PCI PTS HSM 검증 하드웨어 보안 모듈의 분산 플릿의 암호화 작업을 요청하기 위한 여러 인터페이스(AWS CLI, AWS SDK 및를 통한 RESTful API 포함 AWS Management Console)가 포함되어 있습니다.




AWS Payment Cryptography는 웹 기반 AWS Payment Cryptography 호스트와 HSMs. 이러한 계층형 호스트의 그룹화는 AWS Payment Cryptography 스택을 구성합니다. AWS Payment Cryptography에 대한 모든 요청은 전송 계층 보안 프로토콜(TLS)을 통해 이루어져야 하며 AWS Payment Cryptography 호스트에서 종료되어야 합니다. 서비스 호스트는 PFS(Perfect Forward Secrecy)을 제공하는 암호 제품군이 있는 TLS만 허용합니다. 서비스는 다른 모든 AWS API 작업에 사용할 수 있는 IAM의 동일한 자격 증명 및 정책 메커니즘을 사용하여 요청을 인증하고 승인합니다.

AWS Payment Cryptography 서버는 가상이 아닌 프라이빗 네트워크를 통해 기본 HSM에 연결됩니다. 서비스 구성 요소와 HSM 간의 연결은 인증 및 암호화를 위한 상호 TLS(mTLS)로 보호됩니다.

주제

- 설계 목표
- 기본
- 내부 작업

- 고객 작업

# 설계 목표

AWS Payment Cryptography는 다음 요구 사항을 충족하도록 설계되었습니다.

- 신뢰성 - 키 사용은 사용자가 정의하고 관리하는 액세스 제어 정책으로 보호됩니다. 일반 텍스트 AWS Payment Cryptography 키를 내보내는 메커니즘은 없습니다. 암호화 키의 기밀성은 매우 중요합니다. HSM에서 관리 작업을 수행하려면 쿼럼 기반 액세스 제어에 대한 역할별 액세스 권한을 가진 여러 Amazon 직원이 필요합니다. Amazon 직원은 HSM 기본(또는 마스터) 키 또는 백업에 액세스할 수 없습니다. 기본 키는 AWS Payment Cryptography 리전에 속하지 않는 HSMs과 동기화할 수 없습니다. 다른 모든 키는 HSM 기본 키로 보호됩니다. 따라서 고객 AWS Payment Cryptography 키는 고객 계정 내에서 운영되는 AWS Payment Cryptography 서비스 외부에서 사용할 수 없습니다.
- 짧은 지연 시간 및 높은 처리량 - AWS Payment Cryptography는 결제 암호화 키를 관리하고 결제 트랜잭션을 처리하는 데 적합한 지연 시간 및 처리량 수준에서 암호화 작업을 제공합니다.
- 내구성 - 암호화 키의 내구성은 AWS에서 내구성이 가장 높은 서비스의 내구성과 동일하게 설계되었습니다. 단일 암호 키를 결제 단말기, EMV 칩 카드 또는 수년 동안 사용 중인 기타 보안 암호화 장치(SCD)와 공유할 수 있습니다.
- 독립 리전 - AWS는 여러 리전에서 데이터 액세스를 제한해야 하거나 데이터 레지던시 요건을 준수해야 하는 고객을 위해 독립적인 리전을 제공합니다. 키 사용은 단일 AWS 리전으로 제한됩니다.
- 난수의 보안 소스 - 강력한 암호화는 실제로 예측할 수 없는 난수 생성에 의존하기 때문에 AWS Payment Cryptography는 고품질의 검증된 난수 소스를 제공합니다. AWS Payment Cryptography의 모든 키 생성은 PCI 모드에서 작동하는 PCI PTS HSM에 등록된 HSM을 사용합니다.
- 감사 - AWS 결제 암호화는 Amazon CloudWatch를 통해 사용할 수 있는 CloudTrail 로그 및 서비스 로그에 암호화 키의 사용 및 관리를 기록합니다. CloudTrail 로그를 사용하여 키를 공유한 계정의 키 사용을 포함하여 암호화 키 사용을 검사할 수 있습니다. AWS 결제 암호화는 해당 PCI, 카드 브랜드 및 리전 결제 보안 표준에 따라 타사 평가자의 감사를 받습니다. 증명 및 공동 책임 가이드는 AWS Artifact에서 제공됩니다.
- 탄력적 - AWS 결제 암호화는 수요에 따라 스케일 아웃 및 스케일 인됩니다. AWS 결제 암호화는 HSM 용량을 예측하고 예약하는 대신 온디맨드 방식으로 결제 암호화를 제공합니다. AWS Payment Cryptography는 HSM의 보안 및 규정 준수를 유지하여 고객의 최대 수요를 충족할 수 있는 충분한 용량을 제공할 책임이 있습니다.

# 기본

이 장의 주제에서는 AWS Payment Cryptography의 암호화 기본 요소와 사용되는 위치에 대해 설명합니다. 또한 이 서비스의 기본 요소를 소개합니다.

주제


- 암호 프리미티브
- 엔트로피 및 난수 생성
- 대칭 키 작업
- 비대칭 키 작업
- 키 스토리지
- 대칭 키를 사용한 키 가져오기
- 비대칭 키를 사용한 키 가져오기
- 키 내보내기
- 트랜잭션별 파생된 고유 키(DUKPT) 프로토콜
- 키 계층 구조


## 암호 프리미티브

AWS Payment Cryptography는 파라미터가 가능한 표준 암호화 알고리즘을 사용하므로 애플리케이션이 사용 사례에 필요한 알고리즘을 구현할 수 있습니다. 암호화 알고리즘 세트는 PCI, ANSI X9, EMVCo 및 ISO 표준에 따라 정의됩니다. 모든 암호화는 PCI 모드에서 실행되는 PCI PTS HSM 표준 등재 HSM에 의해 수행됩니다.

## 엔트로피 및 난수 생성

AWS Payment Cryptography 키 생성은 AWS Payment Cryptography HSMs에서 수행됩니다. HSM은 지원되는 모든 키 유형 및 파라미터에 대한 PCI PTS HSM 요구 사항을 충족하는 난수 생성기를 구현합니다.

## 대칭 키 작업

ANSI X9 TR 31, ANSI X9.24 및 PCI PIN 부록 C에 정의된 대칭 키 알고리즘 및 키 강도가 지원됩니다.

- 해시 함수 - 출력값 크기가 2551보다 큰 SHA2 및 SHA3 제품군의 알고리즘입니다. PCI PTS POI v3 이전 버전 단말기와의 역호환성은 제외됩니다.

- 암호화 및 암호 해독 - 키 크기가 128비트 이상인 AES 또는 키 크기가 112비트 이상인 TDEA(2 키 또는 3 키)입니다.
- 메시지 인증 코드(MAC) CMAC 또는 GMAC(AES 사용), 승인된 해시 함수가 있고 키 크기가 128보다 크거나 같은 HMAC.

AWS Payment Cryptography는 HSM 기본 키, 데이터 보호 키 및 TLS 세션 키에 AES 256을 사용합니다.

참고: 나열된 일부 함수는 표준 프로토콜 및 데이터 구조를 지원하기 위해 내부적으로 사용됩니다. 특정 작업에서 지원하는 알고리즘은 API 설명서를 참조하세요.

## 비대칭 키 작업

ANSI X9 TR 31, ANSI X9.24 및 PCI PIN 부록 C에 정의된 비대칭 키 알고리즘 및 키 강도가 지원됩니다.

- 승인된 키 설정 체계 - NIST SP800-56A(ECC/FCC2 기반 키 계약), NIST SP800-56B(IFC 기반 키 계약) 및 NIST SP800-38F(AES 기반 키 암호화/래핑)에 설명되어 있습니다.

AWS Payment Cryptography 호스트는 완벽한 순방향 보안을 제공하는 암호 제품군과 함께 TLS를 사용하는 서비스에 대한 연결만 허용합니다.

참고: 나열된 일부 함수는 표준 프로토콜 및 데이터 구조를 지원하기 위해 내부적으로 사용됩니다. 특정 작업에서 지원하는 알고리즘은 API 설명서를 참조하세요.

## 키 스토리지

AWS Payment Cryptography 키는 HSM AES 256 기본 키로 보호되며 암호화된 데이터베이스의 ANSI X9 TR 31 키 블록에 저장됩니다. 데이터베이스는 AWS Payment Cryptography 서버의 인 메모리 데이터베이스에 복제됩니다.

PCI PIN 보안 규범 부록 C에 따르면 AES 256 키는 다음과 같거나 더 강력합니다.

- 3-key TDEA
- RSA 15360비트
- ECC 512비트
- DSA, DH 및 MQV 15360/512

## 대칭 키를 사용한 키 가져오기

AWS Payment Cryptography는 가져오기를 위해 보호된 키와 같거나 더 강력한 대칭 키 암호화 키(KEK)가 있는 대칭 또는 퍼블릭 키가 있는 암호 및 키 블록의 가져오기를 지원합니다.

## 비대칭 키를 사용한 키 가져오기

AWS Payment Cryptography는 가져오기를 위해 보호된 키만큼 강력하거나 강력한 프라이빗 키 암호화 키(KEK)로 보호되는 대칭 또는 퍼블릭 키를 사용하여 암호 및 키 블록 가져오기를 지원합니다. 암호 해독을 위해 제공되는 퍼블릭 키는 고객이 신뢰하는 기관의 인증서를 통해 신뢰성과 무결성을 보장해야 합니다.

AWS Payment Cryptography에서 제공하는 퍼블릭 KEK는 PCI PIN 보안 및 PCI P2PE 부속서 A를 준수하는 것으로 증명된 인증 기관(CA)의 인증 및 무결성 보호를 제공합니다.

## 키 내보내기

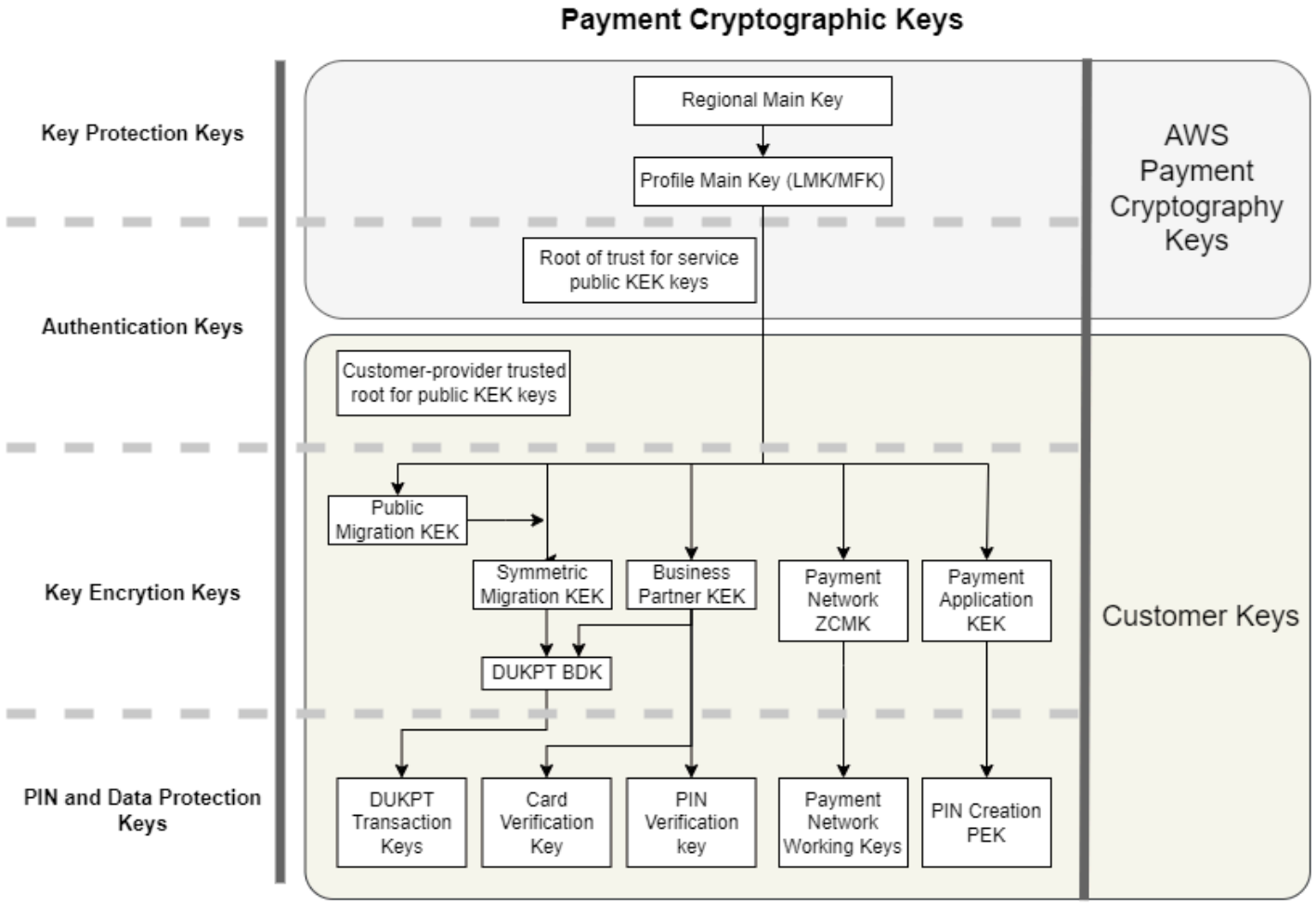
키는 적절한 KeyUsage가 있고 내보낼 키와 같거나 더 강한 키로 내보내고 보호할 수 있습니다.

## 트랜잭션별 파생된 고유 키(DUKPT) 프로토콜

AWS Payment Cryptography는 ANSI X9.24-3에 설명된 대로 TDEA 및 AES 기본 파생 키(BDK)를 지원합니다.

## 키 계층 구조

AWS Payment Cryptography 키 계층 구조는 키가 보호하는 키보다 강력하거나 강력한 키로 키를 항상 보호할 수 있도록 합니다.




AWS Payment Cryptography 키는 서비스 내의 키 보호에 사용됩니다.

| Key(키) | 설명 |
| --- | --- |
| 리전 기본 키 | 암호화 처리에 사용되는 가상 HSM 이미지 또는 프로필을 보호합니다. 이 키는 HSM 및 보안 백업에만 존재합니다. |
| 프로필 기본 키 | 최고 수준 고객 키 보호 키로, 전통적으로 고객 키에 대한 로컬 마스터 키(LMK) 또는 마스터 파일 키(MFK)라고 합니다. 이 키는 HSM 및 보안 백업에만 존재합니다. 프로필은 결제 사용 사례의 보안 표준에서 요구하는 대로 고유한 HSM 구성을 정의합니다. |

| Key(키) | 설명 |
| --- | --- |
| AWS Payment Cryptography 퍼블릭 키 암호화 키(KEK) 키에 대한 신뢰 루트 | 비대칭 키를 사용하여 키 가져오기 및 내보내기를 위해 AWS Payment Cryptography에서 제공하는 퍼블릭 키를 인증하고 검증하기 위한 신뢰할 수 있는 루트 퍼블릭 키 및 인증서입니다. |

고객 키는 다른 키를 보호하는 데 사용되는 키와 결제 관련 데이터를 보호하는 키별로 그룹화됩니다. 두 가지 유형의 고객 키의 예는 다음과 같습니다.

| Key(키) | 설명 |
| --- | --- |
| 공개 KEK 키에 대한 고객이 제공한 신뢰할 수 있는 루트 | 비대칭 키를 사용하여 키를 가져오고 내보낼 때 제공하는 퍼블릭 키를 인증하고 검증하기 위한 신뢰 루트로 사용자가 제공하는 퍼블릭 키 및 인증서입니다. |
| 키 암호화 키(KEK) | KEK는 외부 키 스토어와 AWS Payment Cryptography, 비즈니스 파트너, 결제 네트워크 또는 조직 내 다른 애플리케이션 간의 교환을 위해 다른 키를 암호화하는 데에만 사용됩니다. |
| 트랜잭션별 파생된 고유 키(DUKPT) 기본 파생 키(BDK) | BDK는 각 결제 단말기의 고유 키를 생성하고 여러 단말기의 트랜잭션을 하나의 인수 은행 또는 취득자의 작업 키로 변환하는 데 사용됩니다. PCI Point-to-Point Encryption(P2PE)에서 요구하는 모범 사례는 단말기 모델, 키 삽입 또는 초기화 서비스 또는 기타 세분화에 서로 다른 BDK를 사용하여 BDK 손상의 영향을 제한하는 것입니다. |
| 결제 네트워크 영역 제어 마스터 키(ZCMK) | 영역 키 또는 영역 마스터 키라고도 하는 ZCMK는 결제 네트워크에서 초기 작업 키를 설정하기 위해 제공됩니다. |
| DUKPT 트랜잭션 키 | DUKPT용으로 구성된 결제 단말기는 단말기 및 트랜잭션을 위한 고유한 키를 생성합니다. 트랜 |

| Key(키) | 설명 |
| --- | --- |
| | 잭션을 수신하는 HSM은 단말기 식별자와 거래 시퀀스 번호로부터 키를 결정할 수 있습니다. |
| 카드 데이터 준비 키 | EMV 발급자 마스터 키, EMV 카드 키 및 확인 값, 카드 개인 설정 데이터 파일 보호 키는 카드 개인화 공급자가 사용할 개별 카드에 대한 데이터를 생성하는 데 사용됩니다. 발급 은행 또는 발급자가 트랜잭션 승인의 일환으로 카드 데이터를 인증하는 데에도 이러한 키와 암호 검증 데이터를 사용합니다. |
| 카드 데이터 준비 키 | EMV 발급자 마스터 키, EMV 카드 키 및 확인 값, 카드 개인 설정 데이터 파일 보호 키는 카드 개인화 공급자가 사용할 개별 카드에 대한 데이터를 생성하는 데 사용됩니다. 발급 은행 또는 발급자가 트랜잭션 승인의 일환으로 카드 데이터를 인증하는 데에도 이러한 키와 암호 검증 데이터를 사용합니다. |
| 결제 네트워크 작업 키 | 발급자 작업 키 또는 취득자 작업 키라고도 하는 이 키는 결제 네트워크로 전송되거나 결제 네트워크로부터 수신되는 거래를 암호화하는 키입니다. 이러한 키는 네트워크에서 자주 교체되며, 매일 또는 매시간 교체되는 경우가 많습니다. PIN/Debit 트랜잭션을 위한 PIN 암호화 키(PEK)입니다. |
| 개인 식별 번호(PIN) 암호화 키(PEK) | PIN 블록을 만들거나 해독하는 애플리케이션은 PEK를 사용하여 일반 텍스트 PIN의 저장 또는 전송을 방지합니다. |

# 내부 작업

이 주제에서는 전 세계적으로 분산되고 확장 가능한 Payment Cryptography 및 키 관리 서비스를 위해 고객 키와 암호화 작업을 보호하기 위해 서비스에서 구현하는 내부 요구 사항을 설명합니다.

주제


- HSM 보호
- 일반 키 관리
- 고객 키 관리
- 통신 보안
- 로깅 및 모니터링


# HSM 보호

## HSM 사양 및 수명 주기

AWS Payment Cryptography는 상용 HSMs 플릿을 사용합니다. HSMs은 FIPS 140-2 레벨 3 검증되었으며 PCI 보안 표준 위원회에서 승인한 PCI PTS 디바이스 목록에 PCI HSM v3 준수로 나열된 펌웨어 버전과 보안 정책도 사용합니다. PCI PTS HSM 표준에는 결제 보안 및 규정 준수에 중요하지만 FIPS 140에서 다루지 않는 HSM 하드웨어의 제조, 배송, 배포, 관리 및 폐기에 대한 추가 요구 사항이 포함되어 있습니다.

타사 평가자는 HSM이 모델, 펌웨어, 구성, 수명 주기 물리적 관리, 변경 제어, 운영자 액세스 제어, 기본 키 관리, HSMs 및 P2PE 요구 사항을 생성하는지 확인합니다.

모든 HSM은 PCI 모드에서 작동하며 PCI PTS HSM 보안 정책에 따라 구성됩니다. AWS Payment Cryptography 사용 사례를 지원하는 데 필요한 함수만 활성화됩니다. AWS Payment Cryptography는 일반 텍스트 PINs의 인쇄, 표시 또는 반환을 제공하지 않습니다.

## HSM 디바이스 물리적 보안

배송 전에 제조업체가 AWS Payment Cryptography 인증 기관(CA)에서 서명한 디바이스 키가 있는 HSMs만 서비스에서 사용할 수 있습니다. AWS Payment Cryptography는 HSM 제조업체 및 디바이스 인증서에 대한 신뢰의 루트인 제조업체 CA의 하위 CA입니다. 제조업체의 CA는 PCI PIN 보안 부속서 A 및 PCI P2PE 부속서 A의 준수를 증명했습니다. 제조업체는 AWS Payment Cryptography CA에서 서명한 디바이스 키가 있는 모든 HSM이 AWS의 지정된 수신기로 배송되는지 확인합니다.

PCI PIN 보안에서 요구하는 대로 제조업체는 HSM 배송과 다른 통신 채널을 통해 일련번호 목록을 제공합니다. 이러한 일련 번호는 AWS 데이터 센터에 HSM을 설치하는 프로세스의 각 단계에서 확인됩니다. 마지막으로 AWS Payment Cryptography 연산자는 AWS Payment Cryptography 키를 수신할 수 있는 HSM 목록에 일련 번호를 추가하기 전에 제조업체 목록과 비교하여 설치된 HSM 목록을 검증합니다.

HSM은 항상 안전한 스토리지에 보관되거나 이중 제어 하에 있으며, 여기에는 다음이 포함됩니다.

- 제조업체에서 AWS 랙 조립 시설로 배송.
- 랙 조립 중.
- 랙 조립 시설에서 데이터 센터로 배송.
- 입고 및 데이터 센터 보안 처리실에 설치. HSM 랙은 카드 액세스 제어 잠금 장치, 알람 도어 센서 및 카메라를 통한 이중 제어 기능을 갖추고 있습니다.
- 작동 중.
- 해체 및 폐기 중.

각 HSM는 개별 책임자에 의해 완전한 관리 체계가 유지되고 모니터링됩니다.

## HSM 초기화

HSM은 일련 번호, 제조업체 설치 디바이스 키 및 펌웨어 체크섬으로 자격 증명 및 무결성을 검증한 후에만 AWS Payment Cryptography 플릿의 일부로 초기화됩니다. HSM의 신뢰성과 무결성이 검증된 후에는 PCI 모드 활성화를 포함하여 HSM이 구성됩니다. 그런 다음 AWS Payment Cryptography 리전 기본 키와 프로필 기본 키가 설정되고 서비스에서 HSM을 사용할 수 있습니다.

## HSM 서비스 및 수리

HSM에는 디바이스의 암호화 경계를 위반할 필요가 없는 서비스 가능한 구성 요소가 있습니다. 이러한 구성 요소에는 냉각 팬, 전원 공급 장치 및 배터리가 포함됩니다. HSM 또는 HSM 랙 내의 다른 장치에 서비스가 필요한 경우 랙이 열려 있는 전체 기간 동안 이중 제어가 유지됩니다.

## HSM 해제

서비스 종료는 HSM의 수명 종료 또는 고장으로 인해 발생합니다. HSM이 작동하는 경우 랙에서 제거하기 전에 논리적으로 제로화된 다음 AWS 데이터 센터의 보안 처리실 내에서 폐기됩니다. 이러한 제품은 수리를 위해 제조업체에 반환되거나, 다른 용도로 사용되지 않으며, 또는 파기 전에 안전한 처리실에서 제거되지 않습니다.

## HSM 펌웨어 업데이트

업데이트가 보안과 관련이 있거나 고객이 새 버전의 기능을 활용할 수 있다고 판단되는 경우 PCI PTS HSM 및 FIPS 140-2(또는 FIPS 140-3) 목록 버전과의 정렬을 유지하기 위해 필요한 경우 HSM 펌웨어 업데이트가 적용됩니다. AWS 결제 암호화 HSMs PCI PTS HSM 목록 버전과 일치하는 off-the-shelf 펌

웨어를 실행합니다. 새 펌웨어 버전은 PCI 또는 FIPS 인증 펌웨어 버전과의 무결성을 검증한 다음 모든 HSM에 롤아웃하기 전에 기능 테스트를 거칩니다.

### 운영자 액세스

운영자는 정상적인 운영 중에 HSM에서 수집한 정보만으로는 문제를 식별하거나 변경을 계획하기에 충분하지 않은 경우 콘솔 없이 HSM에 액세스하여 문제를 해결할 수 있습니다. 다음 단계가 실해됩니다.

- 문제 해결 활동이 개발 및 승인되고 비콘솔 세션이 예약됩니다.
- HSM이 고객 처리 서비스에서 제거됩니다.
- 이중 제어 하에 기본 키가 삭제됩니다.
- 운영자는 콘솔 이외에서도 HSM에 액세스하여 이중 제어 상태에서 승인된 문제 해결 활동을 수행할 수 있습니다.
  - 비콘솔이 세션이 종료되면 HSM에서 초기 프로비저닝 프로세스를 수행하여 표준 펌웨어와 구성을 반환한 다음 기본 키를 동기화한 후 HSM을 반환하여 고객에게 서비스합니다.
  - 세션 레코드는 변경 내용 추적에 기록됩니다.
  - 세션에서 얻은 정보는 향후 변경 계획을 수립하는 데 사용됩니다.

콘솔을 이용하지 않은 모든 액세스 기록을 검토하여 프로세스 규정 준수 및 HSM 모니터링, 비콘솔 액세스 관리 프로세스 또는 운영자 교육에 대한 잠재적 변경 사항을 검토합니다.

## 일반 키 관리

리전 내 모든 HSM은 리전 기본 키와 동기화됩니다. 리전 기본 키는 하나 이상의 프로필 기본 키를 보호합니다. 프로필 기본 키는 고객 키를 보호합니다.

모든 기본 키는 HSM에서 생성되며 ANSI X9 TR 34 및 PCI PIN 부록 A에 따라 비대칭 기법을 사용하여 대칭 키 분포를 통해 배포됩니다.

### 생성

AES 256비트 기본 키는 PCI PTS HSM 난수 생성기를 사용하여 서비스 HSM 플릿용으로 프로비저닝된 HSM 중 하나에서 생성됩니다.

### 리전 기본 키 동기화

HSM 리전 기본 키는 ANSI X9 TR-34에 따라 정의한 메커니즘을 사용하여 지역 플릿 전반의 서비스에 의해 동기화되며, 여기에는 다음이 포함됩니다.

- KDH(키 분포 호스트) 와 KRD(키 수신 디바이스) 키 및 인증서를 사용한 상호 인증으로 퍼블릭 키에 대한 인증 및 무결성을 제공합니다.
- 인증서는 PCI PIN Annex A2의 요구 사항을 충족하는 인증 기관(CA)에서 서명합니다. 단, AES 256 비트 키를 보호하는 데 적합한 비대칭 알고리즘과 키 강도는 예외입니다.
- 분산 대칭 키의 식별 및 키 보호는 AES 256비트 키를 보호하는 데 적합한 비대칭 알고리즘 및 키 강도를 제외하고 ANSI X9 TR-34 및 PCI PIN 부속서 A1과 일치합니다.

리전 기본 키는 다음과 같은 방법으로 해당 리전에 대해 인증 및 프로비전된 HSM에 대해 설정됩니다.

- 기본 키는 해당 리전의 HSM에서 생성됩니다. 해당 HSM은 키 분포 호스트로 지정됩니다.
- 해당 리전에 프로비저닝된 모든 HSM은 HSM의 퍼블릭 키와 재생할 수 없는 인증 정보를 포함하는 KRD 인증 토큰을 생성합니다.
- KDH가 HSM의 ID와 키 수신 권한을 확인한 후 KRD 토큰이 KDH 허용 목록에 추가됩니다.
- KDH는 각 HSM에 대해 인증 가능한 기본 키 토큰을 생성합니다. 토큰에는 생성 대상 HSM에서만 로드할 수 있는 KDH 인증 정보와 암호화된 기본 키가 포함되어 있습니다.
- 각 HSM에는 이를 위해 빌드된 기본 키 토큰이 전송됩니다. HSM의 자체 인증 정보와 KDH 인증 정보를 검증한 후 KRD 프라이빗 키로 기본 키를 해독하여 기본 키에 로드합니다.

단일 HSM을 특정 리전과 다시 동기화해야 하는 경우:

- 펌웨어 및 구성을 통해 다시 검증되고 프로비저닝됩니다.
- 해당 리전을 처음 사용하는 경우:
  - HSM은 KRD 인증 토큰을 생성합니다.
  - KDH는 토큰을 허용 목록에 추가합니다.
  - KDH는 HSM의 기본 키 토큰을 생성합니다.
  - HSM은 기본 키를 로드합니다.
  - HSM은 서비스에서 사용할 수 있습니다.

이를 통해 다음이 보장됩니다.

- 리전 내에서 AWS Payment Cryptography 처리에 대해 검증된 HSM만 해당 리전의 마스터 키를 받을 수 있습니다.
- AWS Payment Cryptography HSM의 마스터 키만 플릿의 HSM에 배포할 수 있습니다.

## 리전 기본 키 교체

리전 기본 키는 암호화 기간 만료 시, 키 손상이 의심되는 경우 교체되거나 키 보안에 영향을 미치는 것으로 판단되는 서비스 변경 이후 교체됩니다.

초기 프로비저닝과 마찬가지로 새 리전 기본 키가 생성되고 배포됩니다. 저장된 프로필 기본 키를 새 리전 기본 키로 변환해야 합니다.

리전 기본 키 교체는 고객 처리에 영향을 미치지 않습니다.

## 프로필 기본 키 동기화

프로필 기본 키는 리전 기본 키로 보호됩니다. 이렇게 하면 프로필이 특정 리전으로 제한됩니다.

프로필 기본 키는 다음에 따라 제공됩니다.

- 프로필 기본 키가 리전 기본 키가 동기화된 HSM에서 생성됩니다.
- 프로필 기본 키가 프로필 구성 및 기타 컨텍스트와 함께 저장되고 암호화됩니다.
- 프로필이 리전 기본 키가 있는 리전의 모든 HSM에서 고객 암호화 기능에 사용됩니다.

## 프로필 기본 키 교체

프로필 기본 키는 암호화 기간 만료 시, 키 손상이 의심되는 경우 또는 키 보안에 영향을 미치는 것으로 판단되는 서비스 변경 이후 교체됩니다.

교체 단계:

- 새 프로필 기본 키가 생성되고 초기 프로비저닝과 마찬가지로 보류 중인 기본 키로 배포됩니다.
- 백그라운드 프로세스는 고객 키 자료를 설정된 프로필 기본 키에서 보류 중인 기본 키로 변환합니다.
- 보류 중인 키를 사용하여 모든 고객 키를 암호화하면 보류 중인 키가 프로필 기본 키로 승격됩니다.
- 백그라운드 프로세스를 통해 만료된 키로 보호되는 고객 키 구성 요소가 삭제됩니다.

프로필 기본 키 교체는 고객 처리에 영향을 주지 않습니다.

## 보호

키는 키 계층 구조에만 의존하여 보호됩니다. 모든 고객 키의 손실이나 손상을 방지하려면 기본 키를 보호하는 것이 중요합니다.

리전 기본 키는 백업에서 서비스를 위해 인증되고 프로비저닝된 HSM으로만 복원할 수 있습니다. 이러한 키는 특정 HSM에 대한 특정 KDH의 상호 인증 가능하고 암호화된 기본 키 토큰으로만 저장할 수 있습니다.

프로필 마스터 키는 지역별로 암호화된 프로필 구성 및 컨텍스트 정보와 함께 저장됩니다.

고객 키는 키 블록에 저장되며 프로필 마스터 키로 보호됩니다.

모든 키는 HSM 내에서만 존재하거나 암호화 강도가 같거나 더 강력한 다른 키로 보호되어 저장됩니다.

### 내구성

일반적으로 중단이 발생하는 극단적인 상황에서도 트랜잭션 암호화 및 비즈니스 기능을 위한 고객 키를 사용할 수 있어야 합니다. AWS Payment Cryptography는 가용 영역 및 AWS 리전에 걸쳐 여러 수준의 중복 모델을 활용합니다. 결제 암호화 작업에 대해 서비스에서 제공하는 것보다 더 높은 가용성과 내구성을 원하는 고객은 다중 리전 아키텍처를 구현해야 합니다.

HSM 인증 및 기본 키 토큰은 저장되며 HSM을 재설정해야 하는 경우 기본 키를 복원하거나 새 기본 키와 동기화하는 데 사용할 수 있습니다. 토큰은 필요할 때만 이중 제어 상태에서 보관 및 사용됩니다.

### HSM 기본 키에 대한 운영자 액세스

기본 키는 서비스에서 관리하는 HSM에만 존재하며 보안 AWS 시설에 보호됩니다. 기본 키는 HSM에서 내보내거나 서비스에서 사용하기 위해 제조업체에서 초기화하지 않은 HSM으로 동기화할 수 없습니다. AWS 연산자는 서비스에서 관리하지 않는 HSM에 로드할 수 있는 어떤 형태의 메인 키도 가져올 수 없습니다.

## 고객 키 관리

에서는 AWS고객 신뢰가 최우선 순위입니다. AWS 계정의 서비스에서 가져오거나 생성하는 키에 대한 모든 제어 권한을 유지합니다. 키에 대한 액세스를 구성할 책임은 사용자에게 있습니다.

AWS Payment Cryptography는 HSMs 사용하고 장기 결제 서비스 공급자와 마찬가지로 고객을 대신하여 키를 관리하는 서비스 공급자입니다. 서비스는 HSM 물리적 및 논리적 보안에 대한 완전한 책임이 있습니다. 고객이 서비스에서 생성하거나 서비스로 가져온 키에 대한 정확한 정보를 제공해야 하므로 서비스와 고객 간에 키 관리 책임이 공유됩니다.이 정보는 서비스가 올바른 키 사용 및 관리를 적용하는 데 사용됩니다. AWS 데이터 분리 보호는 한 AWS 계정에 속한 키의 손상으로 인해 다른 계정에 속한 키가 손상되지 않도록 하는 데 사용됩니다.

AWS Payment Cryptography는 서비스에서 관리하는 키의 HSM 물리적 규정 준수 및 키 관리에 대한 모든 책임이 있습니다. 이를 위해서는 HSM 기본 키의 소유권 및 관리와 AWS Payment Cryptography에서 관리하는 고객 키의 보호가 필요합니다.

## 고객 키 스페이스 분리

AWS Payment Cryptography는 키가 다른 계정과 명시적으로 공유되지 않는 한 보안 주체를 키를 소유한 계정으로 제한하는 등 모든 키 사용에 대해 키 정책을 적용합니다.

AWS 계정은 서로 다른 데이터 센터의 비클라우드 구현과 유사한 고객 또는 애플리케이션 간에 완전한 환경 분리를 제공합니다. 각 계정은 격리된 액세스 제어, 네트워킹, 컴퓨팅 리소스, 데이터 스토리지, 데이터 보호 및 결제 트랜잭션을 위한 암호화 키, 모든 AWS 리소스를 제공합니다. Organizations 및 Control Tower와 같은 AWS 서비스를 사용하면 엔터프라이즈 데이터 센터 내의 사육장 또는 방과 유사한 별도의 애플리케이션 계정을 엔터프라이즈에서 관리할 수 있습니다.

## 고객 키에 대한 운영자 액세스

서비스에서 관리하는 고객 키는 파티션 기본 키로 보호되며 소유자가 키 공유를 위해 특별히 구성한 소유 고객 계정 또는 계정에서만 사용할 수 있습니다. AWS 운영자는 AWS 수동 운영자 액세스 메커니즘에서 관리하는 서비스에 대한 수동 액세스를 사용하여 고객 키를 사용하여 키 관리 또는 암호화 작업을 내보내거나 수행할 수 없습니다.

고객 키 관리 및 사용을 구현하는 서비스 코드에는 AWS PCI DSS 평가에 따라 평가된 AWS 보안 코드 관행이 적용됩니다.

## 백업 및 복구

리전의 서비스에 의해 내부적으로 저장된 키 및 키 정보는에서 암호화된 아카이브에 백업됩니다 AWS. 아카이브를 복원하려면의 이중 제어 AWS 가 필요합니다.

## 키 블록

모든 키는 ANSI X9.143 형식 키 블록에 저장되고 처리됩니다.

ImportKey에서 지원하는 암호 또는 기타 키 블록 형식에서 키를 서비스로 가져올 수 있습니다. 마찬가지로 키를 내보낼 수 있는 경우 키 내보내기 프로필에서 지원하는 다른 키 블록 형식 또는 암호문으로 내보낼 수 있습니다.

## 키 사용

키 사용은 서비스에 의해 구성된 KeyUsage로 제한됩니다. 요청된 암호화 작업에 대한 부적절한 키 사용, 사용 모드 또는 알고리즘이 있는 요청은 서비스가 실패합니다.

### 키 교환 관계

PCI PIN 보안 및 PCI P2PE를 사용하려면 해당 키를 공유하는 데 사용되는 키 교환 키(KEK)를 포함하여 PINs 또는 카드 데이터를 암호화하는 키를 공유하는 조직이 다른 조직과 동일한 키를 공유하지 않아야 합니다. 동일한 조직 내를 포함하여 단일 목적으로 두 당사자 간에만 대칭 키를 공유하는 것이 모범 사례입니다. 이렇게 하면 키 손상이 의심되어 영향을 받은 키를 교체해야 하는 등의 영향을 최소화할 수 있습니다.

두 명 이상의 당사자 간에 키를 공유해야 하는 비즈니스 케이스라도 당사자 수를 최소한으로 유지해야 합니다.

AWS Payment Cryptography는 이러한 요구 사항 내에서 키 사용을 추적하고 적용하는 데 사용할 수 있는 키 태그를 제공합니다.

예를 들어, 서비스 공급자와 공유하는 모든 키에 대해 “KIF”=“POSStation”으로 설정하여 다양한 키 입력 기능에 대한 KEK 및 BDK를 식별할 수 있습니다. 또 다른 예로 결제 네트워크와 공유된 키에 “Network”=“PayCard”라는 태그를 붙이는 경우를 생각할 수 있습니다. 태그를 사용하면 액세스 제어를 생성하고 감사 보고서를 생성하여 키 관리 방식을 적용하고 입증할 수 있습니다.

### 키 삭제

DeleteKey는 고객이 구성할 수 있는 기간이 지나면 데이터베이스의 키를 삭제하도록 표시합니다. 이 기간이 지나면 키는 복구할 수 없이 삭제됩니다. 이는 실수 또는 악의적으로 키를 삭제하는 것을 방지하기 위한 안전 메커니즘입니다. 삭제 표시된 키는 RestoreKey를 제외한 어떤 작업에도 사용할 수 없습니다.

삭제된 키는 삭제 후 7일 동안 서비스 백업에 남아 있습니다. 이 기간 동안에는 복구할 수 없습니다.

폐쇄된 AWS 계정에 속한 키는 삭제하도록 표시됩니다. 삭제 기간이 되기 전에 계정을 다시 활성화하면 삭제 표시된 모든 키는 복원되지만 비활성화됩니다. 암호화 작업에 사용하려면 사용자가 다시 활성화해야 합니다.

## 통신 보안

### 외부

AWS Payment Cryptography API 엔드포인트는 요청의 인증 및 무결성을 위해 1.2 이상의 TLS 및 서명 버전 4를 포함한 AWS 보안 표준을 충족합니다.

수신 TLS 연결은 네트워크 로드 밸런서에서 종료되고 내부 TLS 연결을 통해 API 핸들러로 전달됩니다.

### Internal

서비스 구성 요소 간 그리고 서비스 구성 요소와 다른 AWS 서비스 간의 내부 통신은 강력한 암호화를 사용하는 TLS로 보호됩니다.

HSM은 서비스 구성 요소에서만 연결할 수 있는 비 가상 사설 네트워크에 있습니다. HSM과 서비스 구성 요소 간의 모든 연결은 TLS 1.2 이상의 상호 TLS(mTLS)로 보호됩니다. TLS 및 MTL용 내부 인증서는 AWS 프라이빗 인증 기관을 사용하는 Amazon 인증서 관리자에서 관리합니다. 내부 VPCs 및 HSM 네트워크는 예상치 못한 활동 및 구성 변경에 대해 모니터링됩니다.

## 로깅 및 모니터링

내부 서비스 로그에는 다음이 포함됩니다.

- 서비스에서 이루어진 AWS 서비스 호출의 CloudTrail 로그
- CloudWatch 로그에 직접 기록된 이벤트 또는 HSM의 이벤트의 CloudWatch 로그
- HSM 및 서비스 시스템의 로그 파일
- 로그 보관

모든 로그 소스는 키 정보를 포함한 민감한 정보를 모니터링하고 필터링합니다. 로그를 체계적으로 검토하여 민감한 고객 정보가 포함되어 있지 않은지 확인합니다.

직무 수행에 필요한 개인만 로그에 액세스할 수 있습니다.

모든 로그는 AWS 로그 보존 정책에 따라 보관됩니다.

## 고객 작업

AWS Payment Cryptography는 PCI 표준에 따른 HSM 물리적 규정 준수에 대한 모든 책임이 있습니다. 또한이 서비스는 보안 키 스토어를 제공하며 키가 PCI 표준에서 허용하고 생성 또는 가져오는 동안 사용자가 지정한 용도로만 사용될 수 있도록 합니다. 서비스의 보안 및 규정 준수 기능을 활용하기 위한 주요 속성 및 액세스 권한을 구성하는 것은 사용자의 책임입니다.

주제

- 키 생성
- 키 가져오기
- 키 내보내기

- 키 삭제
- 키 교체 중

## 키 생성

키를 생성할 때 서비스가 규정을 준수하여 키를 사용하도록 강제하는 데 사용하는 속성을 설정합니다.

- 알고리즘 및 키 길이
- 사용법
- 가용성 및 만료

ABAC(속성 기반 액세스 제어)에 사용되는 태그를 사용하여 특정 파트너나 애플리케이션에서 사용할 수 있도록 키를 제한하는 데 사용되는 태그도 생성 중에 설정해야 합니다. 태그를 삭제하거나 변경할 수 있는 역할을 제한하는 정책을 포함해야 합니다.

키를 생성하기 전에 키를 사용하고 관리할 수 있는 역할을 결정하는 정책을 설정해야 합니다.

> **Note**
>
> CreateKey 명령의 IAM 정책을 사용하여 키 생성에 대한 이중 제어를 적용하고 시연할 수 있습니다.

## 키 가져오기

키를 가져올 때 키 블록에 암호적으로 바인딩된 정보를 사용하여 서비스에서 키 사용의 규정 준수를 강제하는 속성을 설정합니다. 기본 키 컨텍스트를 설정하는 메커니즘은 소스 HSM으로 생성되고 공유 또는 비대칭 KEK로 보호되는 키 블록을 사용하는 것입니다. 이는 PCI PIN 요구 사항에 부합하며 소스 애플리케이션의 사용량, 알고리즘 및 키 강도를 보존합니다.

가져올 때 키 블록의 정보 외에도 중요한 주요 속성, 태그 및 액세스 제어 정책을 설정해야 합니다.

암호화를 사용하여 키를 가져오는 경우 소스 애플리케이션의 주요 속성이 전송되지 않습니다. 이 메커니즘을 사용하여 속성을 적절하게 설정해야 합니다.

키 관리자가 전송하는 일반 텍스트 구성 요소를 사용하여 키를 교환한 다음 보안실에서 이중 제어를 구현하는 과정을 통해 키를 로드하는 경우가 많습니다. AWS Payment Cryptography에서는 직접 지원되지 않습니다. API는 자체 HSM에서 가져올 수 있는 인증서와 함께 퍼블릭 키를 내보내 서비스에서 가져

올 수 있는 키 블록을 내보냅니다. 이를 통해 자체 HSM을 사용하여 일반 텍스트 구성 요소를 로드할 수 있습니다.

키 검사 값(KCV)을 사용하여 가져온 키가 소스 키와 일치하는지 확인해야 합니다.

ImportKey API의 IAM 정책을 사용하여 키 가져오기에 대한 이중 제어를 시행하고 시연할 수 있습니다.

## 키 내보내기

파트너 또는 온프레미스 애플리케이션과 키를 공유하려면 키를 내보내야 할 수 있습니다. 내보내기에 키 블록을 사용하면 암호화된 키 구성 요소와 함께 기본적인 키 컨텍스트를 유지할 수 있습니다.

키 태그를 사용하여 동일한 태그와 값을 공유하는 키를 KEK로 내보내도록 제한할 수 있습니다.

AWS Payment Cryptography는 일반 텍스트 키 구성 요소를 제공하거나 표시하지 않습니다. 이를 위해서는 키 관리자가 PCI PTS HSM 또는 ISO 13491 테스트를 거친 보안 암호화 디바이스(SCD)에 직접 액세스하여 표시하거나 인쇄해야 합니다. SCD에 비대칭 KEK 또는 대칭 KEK를 설정하여 이중 제어 하에 일반 텍스트 키 구성 요소 생성 과정을 수행할 수 있습니다.

대상 HSM에서 가져온 항목이 소스 키와 일치하는지 확인하려면 키 검사 값(KCV)을 사용해야 합니다.

## 키 삭제

키 삭제 API를 사용하여 구성한 기간이 지난 후 키가 삭제되도록 스케줄을 지정할 수 있습니다. 그 전에는 키를 복구할 수 있습니다. 키가 삭제되면 서비스에서 영구적으로 제거됩니다.

DeleteKey API의 IAM 정책을 사용하여 키 삭제에 대한 이중 제어를 적용하고 시연할 수 있습니다.

## 키 교체 중

키 별칭을 사용하여 새 키를 생성하거나 가져온 다음 새 키를 참조하도록 키 별칭을 수정하여 키 교체의 효과를 구현할 수 있습니다. 이전 키는 관리 방식에 따라 삭제되거나 비활성화될 수 있습니다.

# 에 대한 할당량 AWS Payment Cryptography

AWS 계정에는 각 AWS 서비스에 대한 기본 할당량(이전에는 제한이라고 함)이 있습니다. 다르게 표시되지 않는 한, 리전별로 각 할당량이 적용됩니다. 일부 할당량에 대한 증가를 요청할 수 있으며 다른 할당량은 늘릴 수 없습니다.

| 명칭 | 기본값 | 조정 가능 | 설명 |
| --- | --- | --- | --- |
| 에일리어스 | 지원되는 각 리전: 2,000 | 예 | 현재 리전에서 이 계정에 사용할 수 있는 별칭의 최대 수입니다. |
| 컨트롤 플레인 요청의 합산 비율 | 각각 지원되는 리전: 초당 5개 | 예 | 현재 리전의 이 계정에서 수행할 수 있는 초당 컨트롤 플레인 요청의 최대 수입니다. 이 할당량은 모든 컨트롤 플레인 작업의 합산에 적용됩니다. |
| 데이터 영역 요청의 합산 비율(비대칭) | 각각 지원되는 리전: 초당 20개 | 예 | 현재 리전의 이 계정에서 수행할 수 있는 비대칭 키를 사용하는 데이터 영역 작업에 대한 초당 최대 요청 수입니다. 이 할당량은 모든 데이터 영역 작업의 합산에 적용됩니다. |
| 데이터 영역 요청의 합산 비율(대칭) | 지원되는 각 리전: 초당 500개 | 예 | 현재 리전의 이 계정에서 수행할 수 있는 대칭 키를 사용하는 데이터 영역 작업에 대한 초당 최대 요청 수입니다. 이 할당량은 모든 데이터 영역 작업의 합산에 적용됩니다. |

| 명칭 | 기본값 | 조정 가능 | 설명 |
| --- | --- | --- | --- |
| 키 | 지원되는 각 리전: 2,000 | 예 | 현재 지역에 있는 이 계정의 최대 키 수입니다. 삭제된 키는 제외됩니다. |

# AWS Payment Cryptography 사용 설명서의 문서 기록

다음 표에서는 AWS Payment Cryptography에 대한 설명서 릴리스를 설명합니다.

| 변경 사항 | 설명 | 날짜 |
| --- | --- | --- |
| 새로운 기능 - AS2805 | AS2805 리전 지원을 지원하는 알고리즘 및 흐름 지원 | 2025년 12월 17일 |
| 새로운 기능 - 다중 리전 키 복제 | 다중 리전 키 복제를 사용하면 AWS Payment Cryptography 키를 여러에 복제할 수 있습니다 AWS 리전. | 2025년 9월 10일 |
| 새로운 기능 - ECDH | 이번 릴리스에서는 ECDH를 사용하여 추가 키 교환을 위한 공유 KEK를 설정할 수 있습니다. | 2025년 3월 30일 |
| 새로운 키 교환 지침 | 키 교환에 대한 새로운 지침이 제공됩니다. 일반적인 JCB 명령에 대한 정보도 추가되었습니다. | 2025년 1월 31일 |
| 새 리전 시작 | 유럽(프랑크푸르트), 유럽(아일랜드), 아시아 태평양(싱가포르), 아시아 태평양(도쿄)에서 새로운 리전 출시를 위한 엔드포인트 추가 | 2024년 7월 31일 |
| 데이터 영역 및 동적 키용 CloudTrail | 예제를 포함하여 데이터 영역(암호화) 작업에 CloudTrail을 사용하는 방법에 대한 정보가 추가되었습니다. 또한 AWS Payment Cryptography로 가져와서는 안 되는 일회성 또는 제한적 사용 키를 더 잘 지원하기 | 2024년 7월 10일 |

| | | |
|---|---|---|
| | 위해 특정 함수에 동적 키를 사용하는 방법에 대한 정보 추가 | |
| 업데이트된 예 | 카드 발급에 대한 새 예제 추가 | 2024년 7월 1일 |
| 기능 릴리스 | VPC 엔드포인트(PrivateLink) 및 iCVV 예제에 대한 정보 추가. | 2024년 5월 30일 |
| 기능 릴리스 | RSA를 사용한 키 가져오기/내보내기 및 DUKPT IPEK/IK 키 내보내기와 관련된 새로운 기능에 대한 정보가 추가되었습니다. | 2024년 1월 15일 |
| 최초 릴리스 | AWS Payment Cryptography 사용 설명서의 최초 릴리스 | 2023년 6월 8일 |

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.