



User Guide

AWS Payment Cryptography



AWS Payment Cryptography: User Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Payment Cryptography?	1
Concepts	2
Industry terminology	4
Common key types	4
Other terms	6
Related services	10
For more information	11
Endpoints	11
Control plane endpoints	11
Data plane endpoints	12
Getting started	14
Prerequisites	14
Step 1: Create a key	15
Step 2: Generate a CVV2 value using the key	16
Step 3: Verify the value generated in step 2	16
Step 4: Perform a negative test	17
Step 5: (Optional) Clean up	17
Managing keys	19
Generating keys	19
Generating a 2KEY TDES key	20
Generating a Pin Encryption Key	21
Create an asymmetric (RSA) key	22
Generating a PIN Verification Value (PVV) Key	23
Generating an asymmetric ECC key	24
List keys	25
Enabling and disabling keys	26
Start key usage	27
Stop key usage	28
Deleting keys	29
About the waiting period	30
Import and export keys	33
Import keys	34
Export keys	44
Using aliases	56

About aliases	57
Using aliases in your applications	60
Related APIs	61
Get keys	61
Get the public key/certificate associated with a key pair	62
Tagging keys	63
About tags in AWS Payment Cryptography	63
Viewing key tags in the console	65
Managing key tags with API operations	65
Controlling access to tags	68
Using tags to control access to keys	71
Understanding key attributes	75
Symmetric Keys	75
Asymmetric Keys	77
Data operations	78
Encrypt, Decrypt and Re-encrypt data	78
Encrypt data	79
Decrypt data	83
Generate and verify card data	86
Generate card data	87
Verify card data	88
Generate, translate and verify PIN data	90
Translate PIN data	90
Generate PIN data	92
Verify PIN data	95
Verify auth request (ARQC) cryptogram	97
Building transaction data	98
Transaction data padding	98
Examples	99
Generate and verify MAC	100
Generate MAC	101
Verify MAC	102
Key types for specific data operations	103
GenerateCardData	104
VerifyCardData	105
GeneratePinData (for VISA/ABA schemes)	106

GeneratePinData (for IBM3624)	107
VerifyPinData (for VISA/ABA schemes)	108
VerifyPinData (for IBM3624)	109
Decrypt Data	110
Encrypt Data	111
Translate Pin Data	112
Generate/Verify MAC	113
VerifyAuthRequestCryptogram	114
Import/Export Key	114
Unused key types	115
Common use cases	116
Issuers and issuer processors	116
General Functions	116
Network specific functions	133
Acquiring and payment facilitators	147
Using Dynamic Keys	148
Security	151
Data protection	151
Protecting key material	153
Data encryption	153
Encryption at rest	153
Encryption in transit	153
Internetwork traffic privacy	154
Resilience	154
Regional isolation	155
Multi-tenant design	155
Infrastructure security	156
Isolation of physical hosts	156
Use Amazon VPC and AWS PrivateLink	157
Considerations for AWS Payment Cryptography VPC endpoints	157
Creating a VPC endpoint for AWS Payment Cryptography	158
Connecting to a VPC endpoint	159
Controlling access to a VPC endpoint	159
Using a VPC endpoint in a policy statement	163
Logging your VPC endpoint	166
Security best practices	168

Compliance validation	171
Identity and access management	172
Audience	172
Authenticating with identities	173
AWS account root user	173
IAM users and groups	174
IAM roles	174
Managing access using policies	176
Identity-based policies	176
Resource-based policies	177
Access control lists (ACLs)	177
Other policy types	177
Multiple policy types	178
How AWS Payment Cryptography works with IAM	178
AWS Payment Cryptography Identity-based policies	179
Authorization based on AWS Payment Cryptography tags	181
Identity-based policy examples	181
Policy best practices	181
Using the console	182
Allow users to view their own permissions	183
Ability to access all aspects of AWS Payment Cryptography	184
Ability to call APIs using specified keys	184
Ability to specifically deny a resource	185
Troubleshooting	186
Monitoring	187
CloudTrail logs	187
.....	187
AWS Payment Cryptography information in CloudTrail	188
Control plane events in CloudTrail	189
Data events in CloudTrail	189
Understanding AWS Payment Cryptography Control Plane log file entries	190
Understanding AWS Payment Cryptography Data Plane log file entries	193
Cryptographic details	196
Design goals	197
Foundations	198
Cryptographic primitives	198

Entropy and random number generation	198
Symmetric key operations	198
Asymmetric key operations	199
Key storage	199
Key import using symmetric keys	200
Key import using asymmetric keys	200
Key export	200
Derived Unique Key Per Transaction (DUKPT) protocol	200
Key hierarchy	200
Internal operations	204
HSM specifications and lifecycle	204
HSM device physical security	205
HSM initialization	205
HSM service and repair	206
HSM decommissioning	206
HSM firmware update	206
Operator access	206
Key management	207
Customer operations	213
Generating keys	214
Importing keys	214
Exporting keys	215
Deleting keys	215
Rotating keys	215
Quotas	216
Document history	218

What is AWS Payment Cryptography?

AWS Payment Cryptography is a managed AWS service that provides access to cryptographic functions and key management used in payment processing in accordance with payment card industry (PCI) standards without the need for you to procure dedicated payment HSM instances. AWS Payment Cryptography provides customers performing payment functions such as acquirers, payment facilitators, networks, switches, processors, and banks with the ability to move their payment cryptographic operations closer to applications in the cloud and minimize dependencies on auxiliary data centers or colocation facilities containing dedicated payment HSMs.

The service is designed to meet applicable industry rules including PCI PIN, PCI P2PE, and PCI DSS, and the service leverages hardware that it is [PCI PTS HSM V3 and FIPS 140-2 Level 3 certified](#). It is designed to support low latency and [high levels of up-time and resilience](#). AWS Payment Cryptography is fully elastic and eliminates many of the operational requirements of on premises HSMs, such as the need to provision hardware, securely manage key material, and to maintain emergency backups in secure facilities. AWS Payment Cryptography also provides you with the option to share keys with your partners electronically, eliminating the need to share paper clear text components.

You can use the [AWS Payment Cryptography Control Plane API](#) to create and manage keys.

You can use the [AWS Payment Cryptography Data Plane API](#) to use encryption keys for payment-related transaction processing and associated cryptographic operations.

AWS Payment Cryptography provides important features that you can use to manage your keys:

- Create and manage symmetric and asymmetric AWS Payment Cryptography keys, including TDES, AES, and RSA keys and specify their intended purpose such as for CVV generation or DUKPT key derivation.
- Automatically store your AWS Payment Cryptography keys securely, protected by hardware security modules (HSMs) while enforcing key separation between use cases.
- Create, delete, list, and update aliases, which are "friendly names" that can be used to access or control access to your AWS Payment Cryptography keys.
- Tag your AWS Payment Cryptography keys for identification, grouping, automation, access control, and cost tracking.

- Import and export symmetric keys between AWS Payment Cryptography and your HSM (or 3rd parties) using Key Encryption Keys (KEK) following TR-31(Interoperable Secure Key Exchange Key Block Specification).
- Import and export symmetric Key Encryption Keys (KEK) between AWS Payment Cryptography and other systems using asymmetric key pairs following by using electronic means such as TR-34 (Method For Distribution Of Symmetric Keys Using Asymmetric Techniques).

You can use your AWS Payment Cryptography keys in cryptographic operations, such as:

- Encrypt, decrypt, and re-encrypt data with symmetric or asymmetric AWS Payment Cryptography keys.
- Securely translate sensitive data (such as cardholder pins) between encryption keys without exposing the clear text in accordance with PCI PIN rules.
- Generate or validate cardholder data such as CVV, CVV2 or ARQC.
- Generate and validate cardholder pins.
- Generate or validate MAC signatures.

Concepts

Learn the basic terms and concepts used in AWS Payment Cryptography and how you can use them to help you protect your data.

Alias

A user-friendly name that is associated with an AWS Payment Cryptography key. The alias can be used interchangeably with [key ARN](#) in many of the AWS Payment Cryptography API operations. Aliases allow keys to be rotated or otherwise changed without impacting your application code. The alias name is a string of up to 256 characters. It uniquely identifies an associated AWS Payment Cryptography key within an account and region. In AWS Payment Cryptography, alias names always begin with `alias/`.

The format of an alias name is as follows:

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

For example:

```
alias/sampleAlias2
```

Key ARN

The key ARN is the Amazon Resource Name (ARN) of a key entry in AWS Payment Cryptography. It is a unique, fully qualified identifier for the AWS Payment Cryptography key. A key ARN includes an AWS account, region, and a randomly generated ID. The ARN is not related or derived from the key material. As they are automatically assigned during create or import operations, these values are not idempotent. Importing the same key multiple times will result in multiple key ARNs with their own lifecycle.

The format of a key ARN is as follows:

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

The following is a sample key ARN:

```
arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaiif1lw2h
```

Key Identifier

A Key Identifier is a reference to a key and one (or more) of them are typical inputs to AWS Payment Cryptography operations. Valid key identifiers could be either a [Key Arn](#) a [Key Alias](#).

AWS Payment Cryptography keys


AWS Payment Cryptography keys (keys) are used for all cryptographic functions. Keys are either generated directly by you using the create key command or added to the system by you calling key import. The origin of a key can be determined by reviewing the attribute KeyOrigin. AWS Payment Cryptography also supports derived or intermediate keys used during cryptographic operations such as those used by DUKPT.

These keys have both immutable and mutable attributes defined at creation. Attributes, such as algorithm, length, and usage are defined at creation and cannot be changed. Others, such as effective date or expiration date, can be modified. See the [AWS Payment Cryptography API Reference](#) for a complete list of AWS Payment Cryptography Key attributes.

AWS Payment Cryptography keys have key types, principally defined by [ANSI X9 TR 31](#), that restrict their use to their intended purpose as specified in PCI PIN v3.1 Requirement 19.

Attributes are bound to keys using key blocks when stored, shared with other accounts, or exported as specified in PCI PIN v3.1 Requirement 18-3.

Keys are identified in the AWS Payment Cryptography platform using a unique value known as a key Amazon Resource Name (ARN).

 **Note**

Key ARN is generated when a key is initially created or imported into the AWS Payment Cryptography service. Thus, if adding the same key material multiple times using the import key functionality, the same key material will be located under multiple key but each with a different key lifecycle.

Industry terminology

Topics

- [Common key types](#)
- [Other terms](#)

Common key types

AWK

An acquirer working key (AWK) is a key typically used to exchange data between an acquirer/ acquirer processor and a network (such as Visa or Mastercard). Historically AWK leverages 3DES for encryption and would be represented as `TR31_P0_PIN_ENCRYPTION_KEY`.

BDK

A base derivation key (BDK) is a working key used to derive subsequent keys and is commonly used as part of PCI PIN and PCI P2PE DUKPT process. It is denoted as `TR31_B0_BASE_DERIVATION_KEY`.

CMK

A card master key (CMK) is one or more card specific key(s) typically derived from a [Issuer Master Key](#), *PAN* and *PSN* and are typically 3DES keys. These keys are stored on the EMV Chip during personalization. Examples of CMKs include AC, SMI and SMC keys.

CMK-AC

An application cryptogram (AC) key is used as part of EMV transactions to generate the transaction cryptogram and is a type of [card master key](#).

CMK-SMI

An secure messaging integrity (SMI) key is used as part of EMV to verify the integrity of payloads sent to the card using MAC such as pin update scripts. It is a type of [card master key](#).

CMK-SMC

An secure messaging confidentiality (SMC) key is used as part of EMV to encrypt data sent to the card such as pin updates. It is a type of [card master key](#).

CVK

A card verification key (CVK) is a key used for generating CVV, CVV2 and similar values using a defined algorithm as well as validating an input. It is denoted as a *TR31_CO_CARD_VERIFICATION_KEY*.

IMK

An issuer master key (IMK) is a master key used as part of EMV chip card personalization. Typically there will be 3 IMKs - one each for AC (cryptogram), SMI (script master key for integrity/signature), and SMC (script master key for confidentiality/encryption) keys.

IK

An initial key (IK) is the first key used in the DUKPT process and derives from the Base Derivation Key ([BDK](#)). No transactions are processed on this key, but it is used to derive future keys that will be used for transactions. The derivation method for creating an IK was defined in X9.24-1:2017. When an TDES BDK is used, X9.24-1:2009 is the applicable standard and IK is replaced by Initial Pin Encryption Key (IPEK).

IPEK

An initial PIN encryption key (IPEK) is the initial key used in the DUKPT process and derives from the Base Derivation Key ([BDK](#)). No transactions are processed on this key, but it is used to derive future keys that will be used for transactions. IPEK is a misnomer as this key can also be used to derive data encryption and mac keys. The derivation method for creating an IPEK was defined in X9.24-1:2009. When an AES BDK is used, X9.24-1:2017 is the applicable standard and IPEK is replaced by Initial Key ([IK](#)).

IWK

An issuer working key (IWK) is a key typically used to exchange data between an issuer/issuer processor and a network (such as Visa or Mastercard). Historically IWK leverages 3DES for encryption and is represented as *TR31_PO_PIN_ENCRYPTION_KEY*.

KEK

A key encryption key (KEK) is a key used to encrypt other keys either for transmission or storage. Keys meant for protecting other keys typically have a KeyUsage of *TR31_KO_KEY_ENCRYPTION_KEY* according to the [TR-31](#) standard.

PEK

A PIN encryption key (PEK) is a type of working key used for encrypting PINs either for storage or transmission between two parties. IWK and AWK are two examples of specific uses of pin encryption keys. These keys are represented as *TR31_PO_PIN_ENCRYPTION_KEY*.

PGK

PGK (Pin Generation Key) is another name for a [Pin Verification Key](#). It's not actually used to generate pins (which by default are cryptographically random numbers) but instead are used to generate verification values such as PVV.

PVK

A PIN verification key (PVK) is a type of working key used for generating PIN verification values such as PVV. The two most common kinds are *TR31_V1_IBM3624_PIN_VERIFICATION_KEY* used for generating IBM3624 offset values and *TR31_V2_VISA_PIN_VERIFICATION_KEY* used for Visa/ABA verification values. This can also be known as a [Pin Generation Key](#).

Other terms

ARQC

Authorization Request Cryptogram (ARQC) is a cryptogram generated at transaction time by an EMV standard chip card (or equivalent contactless implementation). Typically, an ARQC is generated by a chip card and forwarded to an issuer or their agent to verify at transaction time.

CVV

A card verification value is a static secret value that was traditionally embedded on a magnetic stripe and used to validate the authenticity of a transaction. The algorithm is also used for

other purposes such as iCVV, CAVV, CVV2. It may not be embedded in this way for other use cases.

CVV2

A card verification value 2 is a static secret value that was traditionally printed on the front (or back) of a payment card and is used to verify authenticity for card not present payments (such as on the phone or online). It uses the same algorithm as CVV but the service code is set to 000.

iCVV

iCVV is a CVV2-like value but embedded with the track2 equivalent data on a EMV(Chip) card. This value is calculated using a service code of 999 and is different than the CVV1/CVV2 to prevent stolen information from being used to create new payment credentials of a different type. For instance, if chip transaction data was obtained, it is not possible to use this data to generate a magnetic stripe(CVV1) or for online purchases (CVV2).

It uses a [???](#) key

DUKPT

Derived Unique Key Per Transaction (DUKPT) is a key management standard typically used to define the use of one-time use encryption keys on physical POS/POI. Historically DUKPT leverages 3DES for encryption. The industry standard for DUKPT is defined in ANSI X9.24-3-2017.

EMV

[EMV](#) (originally Europay, Mastercard, Visa) is a technical body that works with payment stakeholders to create interoperable payment standards and technologies. One example standard is for chip/contactless cards and the payment terminals they interact with, including the cryptography used. EMV key derivation refers to method(s) of generating unique keys for each payment card based on an initial set of keys such as an [IMK](#)

HSM

A Hardware Security Module (HSM) is a physical device that protects cryptographic operations (for example, encryption, decryption, and digital signatures) as well as the underlying keys used for these operations.

KCV

Key Check Value (KCV) refers to a variety of checksum methods primary used to compare to keys to each other without having access to the actual key material. KCV have also been used

for integrity validation (especially when exchanging keys), although this role is now included as part of key block formats such as [TR-31](#). For TDES keys, the KCV is computed by encrypting 8 bytes, each with value of zero, with the key to be checked and retaining the 3 highest order bytes of the encrypted result. For AES keys, the KCV is computed using a CMAC algorithm where the input data is 16 bytes of zero and retaining the 3 highest order bytes of the encrypted result.

KDH

A Key Distribution Host (KDH) is a device or system that is sending keys in a key exchange process such as [TR-34](#). When sending keys from AWS Payment Cryptography, it is considered the KDH.

KIF

A Key Injection Facility (KIF) is a secure facility used for initializing payment terminals including loading them with encryption keys.

KRD

A Key Receiving Device (KRD) is a device that is receiving keys in a key exchange process such as [TR-34](#). When sending keys to AWS Payment Cryptography, it is considered the KRD.

KSN

A Key Serial Number (KSN) is a value used as an input to DUKPT encryption/decryption to create unique encryption keys per transaction. The KSN typically consists of a BDK identifier, a semi-unique terminal ID as well as a transaction counter that increments on each transition processed on a given payment terminal.

MPoC

MPoC (Mobile Point of Sale on Commercial hardware) is a PCI standard that addresses the security requirements for solutions that enable merchants to accept cardholder PINs or contactless payments using a smartphone or other commercial off-the-shelf (COTS) mobile devices.

PAN

A Primary Account Number (PAN) is a unique identifier for an account such as a credit or debit card. Typically 13-19 digits in length. The first 6-8 digits identifies the network and the issuing bank.

PIN Block

A block of data containing a PIN during processing or transmission as well as other data elements. PIN block formats standardize the content of the PIN block and how it can be processed to retrieve the PIN. Most PIN block are composed of the PIN, the PIN length, and frequently contain part or all of the PAN. AWS Payment Cryptography supports ISO 9564-1 formats 0, 1, 3 and 4. Format 4 is required for AES keys. When verifying or translating PINs, there is a need to specify the PIN block of the incoming or outgoing data.

POI

Point of Interaction (POI), also frequently used synonymously with Point of Sale (POS), is the hardware device that the cardholder interacts with to present their payment credential. An example of a POI is the physical terminal in a merchant location. For the list of certified PCI PTS POI terminals, see the [PCI website](#).

PSN

PAN Sequence Number (PSN) is a numeric value used to differentiate multiple cards issued with the same [PAN](#).

Public key

When using asymmetric ciphers (RSA), the public key is the public component of a public-private key pair. The public key can be shared and distributed to entities that need to encrypt data for the owner of the public-private key pair. For digital signature operations, the public key is used to verify the signature.

Private key

When using asymmetric ciphers (RSA), the private key is the private component of a public-private key pair. The private key is used to decrypt data or create digital signatures. Similar to symmetric AWS Payment Cryptography keys, private keys are securely created by HSMs. They are decrypted only into the volatile memory of the HSM and only for the time needed to process your cryptographic request.

PVV

A PIN verification value (PVV) is a type of cryptographic output that can be used to verify a pin without storing the actual pin. Although it is a generic term, in the context of AWS Payment Cryptography, PVV refers to the Visa or ABA PVV method. This PVV is a four digit number whose inputs are card number, pan sequence number, the pan itself and a PIN verification key.

During the validation stage, AWS Payment Cryptography internally recreates the PVV using the transaction data and compares it again the value that has been stored by the AWS Payment Cryptography customer. In this way, it is conceptually similar to a cryptographic hash or MAC.

RSA Wrap/Unwrap

RSA wrap uses an asymmetric key to wrap a symmetric key (such as a TDES key) for transmission to another system. Only the system with the matching private key can decrypt the payload and load the symmetric key. Conversely, RSA unwrap, will securely decrypt a key encrypted using RSA and then load the key into the AWS Payment Cryptography. RSA wrap is a low level method of exchanging keys and does not transmit keys in key block format and does not utilize payload signing by the sending party. Alternate controls should be considered to ascertain providence and key attributes are not mutated.

TR-34 also utilizes RSA internally, but is a separate format and is not interoperable.

TR-31

TR-31 (formally defined as ANSI X9 TR 31) is a key block format that is defined by the American National Standards Institute (ANSI) to support defining key attributes in the same data structure as the key data itself. The TR-31 key block format defines a set of key attributes that are tied to the key so that they are held together. AWS Payment Cryptography uses TR-31 standardized terms whenever possible to ensure proper key separation and key purpose. TR-31 has been superseded by [ANSI X9.143-2022](#).

TR-34

TR-34 is an implementation of ANSI X9.24-2 that described a protocol to securely distribute symmetric keys (such as 3DES and AES) using asymmetric techniques (such as RSA). AWS Payment Cryptography uses TR-34 methods to permit secure import and export of keys.

Related services

[AWS Key Management Service](#)

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to create and control the cryptographic keys that are used to protect your data. AWS KMS uses hardware security modules (HSMs) to protect and validate your AWS KMS keys.

[AWS CloudHSM](#)

AWS CloudHSM provides customers with dedicated general purpose HSM instances in the AWS Cloud. AWS CloudHSM can provide a variety of cryptographic functions such as creating keys, data signing or encrypting and decrypting data.

For more information

- To learn about the terms and concepts used in AWS Payment Cryptography, see [AWS Payment Cryptography Concepts](#).
- For information about the AWS Payment Cryptography Control Plane API, see [AWS Payment Cryptography Control Plane API Reference](#).
- For information about the AWS Payment Cryptography Data Plane API, see [AWS Payment Cryptography Data Plane API Reference](#).
- For detailed technical information about how AWS Payment Cryptography uses cryptography and secures AWS Payment Cryptography keys, see [Cryptographic details](#).

Endpoints for AWS Payment Cryptography

To connect programmatically to AWS Payment Cryptography, you use an endpoint, the URL of the entry point for the service. The AWS SDKs and the command line tools automatically use the default endpoint for the service in an AWS Region based on the region context of a request, so there's typically no need to explicitly set these values. When needed, you can specify a different endpoint for your API requests.

Control plane endpoints

Region name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	controlplane.payment-cryptography.us-east-1.amazonaws.com	HTTPS
US East (Ohio)	us-east-2	controlplane.payment-cryptography.us-east-2.amazonaws.com	HTTPS

Region name	Region	Endpoint	Protocol
US West (Oregon)	us-west-2	controlplane.payment-cryptography.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	controlplane.payment-cryptography.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	controlplane.payment-cryptography.ap-northeast-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	controlplane.payment-cryptography.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	controlplane.payment-cryptography.eu-west-1.amazonaws.com	HTTPS

Data plane endpoints

Region name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	dataplane.payment-cryptography.us-east-1.amazonaws.com	HTTPS
US East (Ohio)	us-east-2	dataplane.payment-cryptography.us-east-2.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	dataplane.payment-cryptography.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	dataplane.payment-cryptography.ap-southeast-1.amazonaws.com	HTTPS

Region name	Region	Endpoint	Protocol
Asia Pacific (Tokyo)	ap-northeast-1	dataplane.payment-cryptography.ap-northeast-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	dataplane.payment-cryptography.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	dataplane.payment-cryptography.eu-west-1.amazonaws.com	HTTPS

Getting started with AWS Payment Cryptography

To get started with AWS Payment Cryptography, you'll first want to create keys and then use them in various cryptographic operations. The below tutorial provides a simple use case of generating a key to be used for generating/verifying CVV2 values. To try out other examples and to explore deployment patterns within AWS, please try out the following [AWS Payment Cryptography Workshop](#) or explore our sample project available on [Github](#)

This tutorial walks you through creating a single key and performing cryptographic operations using the key. Afterward, you delete the key if you no longer want it, which completes the key lifecycle.

Warning

Examples throughout this user guide may use sample values. We *strongly recommend* not using sample values in a production environment such as key serial numbers.

Topics

- [Prerequisites](#)
- [Step 1: Create a key](#)
- [Step 2: Generate a CVV2 value using the key](#)
- [Step 3: Verify the value generated in step 2](#)
- [Step 4: Perform a negative test](#)
- [Step 5: \(Optional\) Clean up](#)

Prerequisites

Before you begin, make sure that:

- You have permission to access the service. For more information, see [IAM policies](#).
- You have the [AWS CLI](#) installed. You can also use [AWS SDKs](#) or [AWS APIs](#) to access AWS Payment Cryptography, but the instructions in this tutorial use the AWS CLI.

Step 1: Create a key

The first step is to create a key. For this tutorial, you create a [CVK](#) double-length 3DES (2KEY TDES) key for generating and verifying CVV/CVV2 values.

```
$ aws payment-cryptography create-key --exportable --key-attributes
  KeyAlgorithm=TDDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=ENCRYPT,DECRYPT,WRAP,UNWRAP,GENERATE,SIGN,VERIFY,DERIVEKEY,NORESTRICTIONS
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyAttributes": {
      "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDDES_2KEY",
      "KeyModesOfUse": {
        "Encrypt": false,
        "Decrypt": false,
        "Wrap": false,
        "Unwrap": false,
        "Generate": true,
        "Sign": false,
        "Verify": true,
        "DeriveKey": false,
        "NoRestrictions": false
      }
    },
    "KeyCheckValue": "CADD1",
    "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"
  }
}
```

Take note of the KeyArn that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi`. You need that in the next step.

Step 2: Generate a CVV2 value using the key

In this step, you generate a CVV2 for a given [PAN](#) and expiration date using the key from step 1.

```
$ 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": "CADD1",
  "CardDataGenerationKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/tqv5yij6wtxx64pi",
  "CardDataType": "CARD_VERIFICATION_VALUE_2",
  "CardDataValue": "144"
}
```

Take note of the `cardDataValue`, in this case the 3-digit number 144. You need that in the next step.

Step 3: Verify the value generated in step 2

In this example, you validate the CVV2 from step 2 using the key you created in step 1.

Run the following command to validate the CVV2.

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

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

```
"KeyCheckValue": "CADD1"  
}
```

The service returns an HTTP response of 200 to indicate that it validated the CVV2.

Step 4: Perform a negative test

In this step, you create a negative test where the CVV2 is not correct and does not validate. You attempt to validate an incorrect CVV2 using the key you created in step 1. This is an expected operation for example if a cardholder entered the wrong CVV2 at checkout.

```
$ 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.
```

The service returns an HTTP response of 400 with the message "Card validation data verification failed" and a reason of INVALID_VALIDATION_DATA.

Step 5: (Optional) Clean up

Now you can delete the key you created in step 1. To minimize unrecoverable changes, the default key deletion period is seven days.

```
$ 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": "CADD1",
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
    "KeyState": "DELETE_PENDING",
    "UsageStartTimestamp": "2022-10-27T08:27:51.753000-07:00"
  }
}
```

Take note of two fields in the output. The `deletePendingTimestamp` is set to seven days in the future by default. The `keyState` is set to `DELETE_PENDING`. You can cancel this deletion any time before the scheduled deletion time by calling [restore-key](#).

Managing keys

To get started with AWS Payment Cryptography, you'll want create an AWS Payment Cryptography key.

The topics in this section explain how to create and manage a variety of AWS Payment Cryptography key types, from creation to deletion. It includes topics on creating, editing and viewing keys, tagging keys, creating key aliases, as well as enabling and disabling keys.

Topics

- [Generating keys](#)
- [List keys](#)
- [Enabling and disabling keys](#)
- [Deleting keys](#)
- [Import and export keys](#)
- [Using aliases](#)
- [Get keys](#)
- [Tagging keys](#)
- [Understanding key attributes for AWS Payment Cryptography key](#)

Generating keys

You can create AWS Payment Cryptography keys by using the CreateKey API operation. During this process, you will specify various attributes of the key or the resultant output such as the key algorithm (for example, TDES_3KEY), the KeyUsage (for example TR31_PO_PIN_ENCRYPTION_KEY) , permitted operations (for example, encrypt, signing) and whether it is exportable. You cannot change these properties after the AWS Payment Cryptography key is created.

Examples

- [Generating a 2KEY TDES key](#)
- [Generating a Pin Encryption Key](#)
- [Create an asymmetric \(RSA\) key](#)
- [Generating a PIN Verification Value \(PVV\) Key](#)

- [Generating an asymmetric ECC key](#)

Generating a 2KEY TDES key

Example

This command generates a 2KEY TDES key for the purpose of generating and verifying CVV/CVV2 values. The response echos back the request parameters, including an ARN for subsequent calls as well as a KCV (Key Check Value).

```
$ 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}'
```

```
{
  "Key": {
    "CreateTimestamp": "2022-10-26T16:04:11.642000-07:00",
    "Enabled": true,
    "Exportable": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
hjprdg5o4jtgs5tw",
    "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": "B72F",
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
```

```

    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
    "KeyState": "CREATE_COMPLETE",
    "UsageStartTimestamp": "2022-10-26T16:04:11.559000-07:00"
  }
}

```

Generating a Pin Encryption Key

Example Generating a Pin Encryption Key (PEK)

This command generates a 3KEY TDES key for the purpose of encrypting PIN values (known as a Pin Encryption Key). This key may be used for securing storing PINs or for decrypting PINs provided during a verification attempt, for example during a transaction. The response echos back the request parameters, including an ARN for subsequent calls as well as a KCV (Key Check Value).

```

$ 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}'

```

```

{
  "Key": {
    "CreateTimestamp": "2022-10-27T08:27:51.795000-07:00",
    "Enabled": true,
    "Exportable": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
    kwapwa6qaifllw2h",
    "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": "9CA6",
"KeyCheckValueAlgorithm": "ANSI_X9_24",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"KeyState": "CREATE_COMPLETE",
"UsageStartTimestamp": "2022-10-27T08:27:51.753000-07:00"
}
}

```

Create an asymmetric (RSA) key

Example

In this example, we will generate a new asymmetric RSA 2048 bit key pair. A new private key will be generated as well as the matching public key. The public key can be retrieved using the [getPublicCertificate](#) API.

```

$ 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}'

```

```

{
  "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": "CMAC",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"KeyState": "CREATE_COMPLETE",
"UsageStartTimestamp": "2022-11-15T11:15:42.182000-08:00"
}
}

```

Generating a PIN Verification Value (PVV) Key

Example

This command generates a 3KEY TDES key for the purpose of generating PVV values (known as a Pin Verification Value). You can use this key for generating a PVV value that can be compared against a subsequent calculated PVV. The response echos back the request parameters, including an ARN for subsequent calls as well as a KCV (Key Check Value).

```

$ 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}'

```

```

{
  "Key": {
    "CreateTimestamp": "2022-10-27T10:22:59.668000-07:00",
    "Enabled": true,
    "Exportable": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
j4u4cmnzkelhc6yb",
    "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": "5132",
  "KeyCheckValueAlgorithm": "ANSI_X9_24",
  "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
  "KeyState": "CREATE_COMPLETE",
  "UsageStartTimestamp": "2022-10-27T10:22:59.614000-07:00"
}
}
```

Generating an asymmetric ECC key

Example

This command generates an ECC key pair for the purpose of establishing an ECDH (Elliptic Curve Diffie-Hellman) key agreement between two parties by deriving a shared key from their public-private key pairs. Using ECDH, each party generates its own ECC key pair with key purpose K3 and mode of use X and exchange the public keys. Both parties then use their private key and the received public key to establish a shared derived key.

To maintain the single use principle of cryptographic keys in payments, it is recommended to not re-use ECC key pairs for multiple purposes, such as ECDH key derivation and signing.

```
$ 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}'
```

```
{
  "Key": {
    "CreateTimestamp": "2024-10-17T01:31:55.908000+00:00",
    "Enabled": true,
    "Exportable": true,
    "KeyArn": "arn:aws:payment-cryptography:us-west-2:075556953750:key/
xzydvquw6ejfxnwq",
    "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": "CMAC",
    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
    "KeyState": "CREATE_COMPLETE",
    "UsageStartTimestamp": "2024-10-17T01:31:55.866000+00:00"
  }
}
```

List keys

List Keys presents a list of keys accessible to the caller in this account and Region.

Example

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

```
{"Keys": [
```



```
{
  "CreateTimestamp": "2022-10-12T10:58:28.920000-07:00",
  "Enabled": false,
  "Exportable": true,
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
alsuwxug3pgy6xh",
  "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": "2022-10-27T14:19:42.488000-07:00"
}
]
```

Enabling and disabling keys

You can disable and re-enable AWS Payment Cryptography keys. When you create key, it is enabled by default. If you disable a key, it cannot be used in any [cryptographic operation](#) until you re-enable it. Start/stop usage commands take immediate effect, so it's recommended that you review usage before making such a change. You can also set a change (start or stop usage) to take effect in the future using the optional `timestamp` parameter.

Because it's temporary and easily undone, disabling an AWS Payment Cryptography key is a safer alternative to deleting an AWS Payment Cryptography key, an action that is destructive and

irreversible. If you are considering deleting an AWS Payment Cryptography key, disable it first and ensure that you will not need to use the key to encrypt or decrypt data in the future.

Topics

- [Start key usage](#)
- [Stop key usage](#)

Start key usage

Key usage must be enabled in order to use a key for cryptographic operations. If a key is not enabled, you can use this operation to make it usable. The field `UsageStartTimestamp` will represent when the key became/will become active. This will be in the past for an enabled token, and in the future if pending activation.

Example

In this example, a key is requested to be enabled for key usage. The response includes the key information and the enable flag has been transitioned to true. This will also be reflected in list-keys response object.

```
$ aws payment-cryptography start-key-usage --key-identifier "arn:aws:payment-cryptography:us-east-2:111122223333:key/alsuwxug3pgy6xh"
```

```
{
  "Key": {
    "CreateTimestamp": "2022-10-12T10:58:28.920000-07:00",
    "Enabled": true,
    "Exportable": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/alsuwxug3pgy6xh",
    "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"
}
}

```

Stop key usage

If you no longer plan to use a key, you can stop the key usage to prevent further cryptographic operations. This operation is not permanent, so you are able to reverse it using [starting key usage](#). You can also set a key to be disabled in the future. The field `UsageStopTimestamp` will represent when the key became/will become disabled.

Example

In this example, it's requested to stop key usage in the future. After execution, this key cannot be used for cryptographic operations unless re-enabled via [start key usage](#). The response includes the key information and the enable flag has been transitioned to false. This will also be reflected in list-keys response object.

```
$ 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"
}
}
```

Deleting keys

Deleting an AWS Payment Cryptography key deletes the key material and all metadata associated with the key and is irreversible unless a copy of the key is available outside of AWS Payment Cryptography. After a key is deleted, you can no longer decrypt the data that was encrypted under that key, which means that data may become unrecoverable. You should delete a key only when you are sure that you don't need to use it anymore and no other parties are utilizing this key. If you are not sure, consider disabling the key instead of deleting it. You can re-enable a disabled key if you need to use it again later, but you cannot recover a deleted AWS Payment Cryptography key unless you are able to re-import it from another source.

Before deleting a key, you should ensure that you no longer need the key. AWS Payment Cryptography does not store the results of cryptographic operations like CVV2 and is unable to determine if a key is needed for any persistent cryptographic material.

AWS Payment Cryptography never deletes keys belonging to active AWS accounts unless you explicitly schedule them for deletion and the mandatory waiting period expires.

However, you might choose to delete an AWS Payment Cryptography key for one or more of the following reasons:

- To complete the key lifecycle for a key that you no longer need
- To avoid the management overhead associated with maintaining unused AWS Payment Cryptography keys

Note

If you [close or delete your AWS account](#), your AWS Payment Cryptography key become inaccessible. You do not need to schedule deletion of your AWS Payment Cryptography key separate from closing the account.

AWS Payment Cryptography records an entry in your [AWS CloudTrail](#) log when you schedule deletion of the AWS Payment Cryptography key and when the AWS Payment Cryptography key is actually deleted.

About the waiting period

Because deleting a key is irreversible, AWS Payment Cryptography requires you to set a waiting period of between 3–180 days. The default waiting period is seven days.

However, the actual waiting period might be up to 24 hours longer than the one you scheduled. To get the actual date and time when the AWS Payment Cryptography key will be deleted, use the `GetKey` operations. Be sure to note the time zone.

During the waiting period, the AWS Payment Cryptography key status and key state is **Pending deletion**.

Note

An AWS Payment Cryptography key pending deletion cannot be used in any [cryptographic operations](#).

After the waiting period ends, AWS Payment Cryptography deletes the AWS Payment Cryptography key, its aliases, and all related AWS Payment Cryptography metadata.

Use the waiting period to ensure that you don't need the AWS Payment Cryptography key now or in the future. If you find that you do need the key during the waiting period, you can cancel

key deletion before the waiting period ends. After the waiting period ends, you cannot cancel key deletion, and the service deletes the key.

Example

In this example, a key is requested to be deleted. Besides the basic key information, two relevant fields are that key state has been changed to `DELETE_PENDING` and `deletePendingTimestamp` represents when the key is currently scheduled to delete.

```
$ aws payment-cryptography delete-key \  
    --key-identifier arn:aws:payment-cryptography:us-  
east-2:111122223333:key/kwapwa6qaif1lw2h
```

```
{  
  "Key": {  
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/  
kwapwa6qaif1lw2h",  
    "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": "",  
    "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

In this example, a pending deletion is cancelled. Once completed successfully, a key will no longer be deleted per the previous schedule. The response contains the basic key information; additionally, two relevant fields have changed - `KeyState` and `deletePendingTimestamp`. `KeyState` is returned to a value of `CREATE_COMPLETE`, while `DeletePendingTimestamp` is removed.

```

$ aws payment-cryptography restore-key --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h

```

```

{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaif1lw2h",
    "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": "",
    "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"  
}  
}
```

Import and export keys

AWS Payment Cryptography keys can be imported from other solutions or exported to other solutions (such as other HSMs). It is a common use case to exchange keys with service providers using import and export functionality. As a cloud service, AWS Payment Cryptography takes a modern, electronic approach to key management while helping you maintain applicable compliance and controls. The long-term objective is to move away from paper-based key components towards standards-based, electronic means of key exchange.

Key Encryption Key (KEK) Exchange

AWS Payment Cryptography encourages the use of public key cryptography (RSA) for the initial key exchange using the well established [ANSI X9.24 TR-34](#) norm. Common names for this initial key type includes Key Encryption Key (KEK), Zone Master Key (ZMK) and Zone Control Master Key (ZCMK). If your systems or partners are not yet able to support TR-34, you can also consider utilizing [RSA Wrap/Unwrap](#).

If you have a need to continue processing paper key components until all partners support electronic key exchange, you can consider retaining an offline HSM for this purpose.

Note

If you would like to import your own test keys, please check out the sample project on [Github](#). For instructions on how to import/export keys from other platforms, please consult the user guide for those platforms.

Working Key (WK) Exchange

AWS Payment Cryptography uses the relevant industry norm ([ANSI X9.24 TR 31-2018](#)) for exchanging working keys. TR-31 assumes that a KEK has previously been exchanged. This is consistent with requirement of PCI PIN to cryptographically bind key material to its key type

and usage at all times. Working keys have various names including acquirer working keys, issuer working keys, BDK, IPEK, etc.

Topics

- [Import keys](#)
- [Export keys](#)

Import keys

Important

Examples may require the latest version of the AWS CLI V2. Before getting started, please ensure that you have upgraded to the [latest version](#).

Topics

- [Importing symmetric keys](#)
- [Importing asymmetric \(RSA\) keys](#)

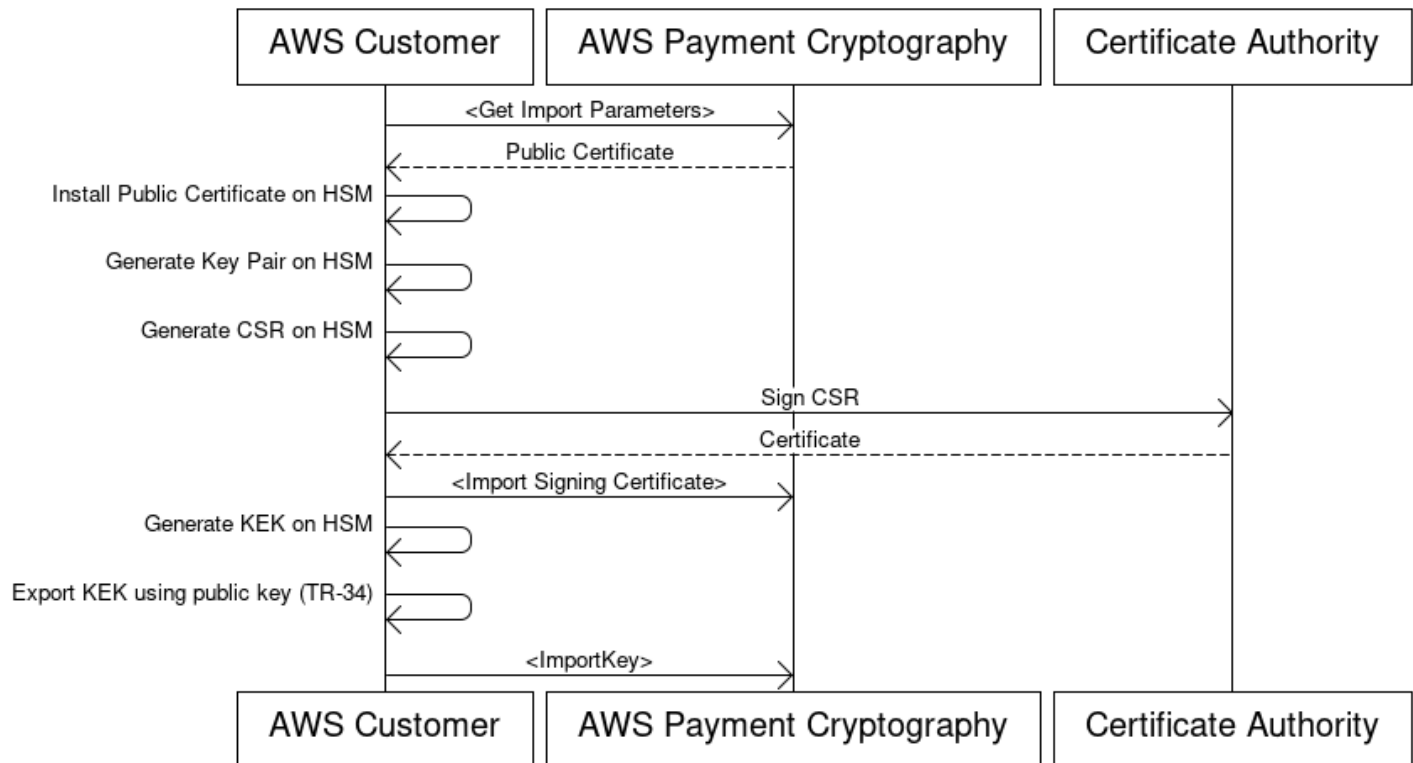
Importing symmetric keys

Topics

- [Import keys using asymmetric techniques \(TR-34\)](#)
- [Import keys using asymmetric techniques \(RSA Unwrap\)](#)
- [Import symmetric keys using a pre-established key exchange key \(TR-31\)](#)

Import keys using asymmetric techniques (TR-34)

Key Encryption Key(KEK) Import Process



Overview: TR-34 utilizes RSA asymmetric cryptography to encrypt symmetric keys for exchange as well as ensuring the source of the data (signing). This ensures both the confidentiality (encryption) and integrity (signature) of the wrapped key.

If you would like to import your own keys, please check out the sample project on [Github](#). For instructions on how to import/export keys from other platforms, please consult the user guide for those platforms.

1. Call the initialize import command

Call `get-parameters-for-import` to initialize the import process. This API will generate a keypair for the purpose of key imports, sign the key and return back the certificate and certificate root. Ultimately, the key to be exported should be encrypted using this key. In TR-34 terminology, this is known as the KRD Cert. Note that these certificates are short lived and are only intended for this purpose.

2. Install public certificate on key source system

With many HSMs, you may need to install/load/trust the public certificate generated in step 1 in order to export keys using it.

3. Generate public key and provide certificate root to AWS Payment Cryptography

To ensure integrity of the transmitted payload, it is signed by the sending party (known as the Key Distribution Host or KDH). The sending party will want to generate a public key for this purpose and then create a public key certificate (X509) that can be provided back to AWS Payment Cryptography. AWS Private CA is one option for generating certificates, but there is no restrictions on the certificate authority used.

Once you have the certificate, you'll want to load the root certificate to AWS Payment Cryptography using the `importKey` command and `KeyMaterialType` of `ROOT_PUBLIC_KEY_CERTIFICATE` and `KeyUsageType` of `TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`.

4. Export key from source system

Many HSMs and related systems support the ability to export keys using the TR-34 norm. You'll want to specify the public key from step 1 as the KRD (encryption) cert and the key from step 3 as the KDH (signing) cert. In order to import to AWS Payment Cryptography, you'll want to specify the format to be TR-34.2012 non-CMS two pass format which may also be referred to as the TR-34 Diebold format.

5. Call import key

As the last step, you will call the `importKey` API with a `KeyMaterialType` of `TR34_KEY_BLOCK`. The `certificate-authority-public-key-identifier` will be the `keyARN` of the root CA imported in step 3, `key-material` will be wrapped key material from step 4 and `signing-key-certificate` is the leaf certificate from step 3. You will also need to provide the `import-token` from step 1.

6. Use imported key for cryptographic operations or subsequent import

If the imported `KeyUsage` was `TR31_K0_KEY_ENCRYPTION_KEY`, then this key can be used for subsequent key imports using TR-31. If the key type was any other type (such as `TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY`), then the key can be directly used for cryptographic operations.

Import keys using asymmetric techniques (RSA Unwrap)

Overview: AWS Payment Cryptography supports RSA wrap/unwrap for key exchange when TR-34 is not feasible. Similar to TR-34, this technique utilizes RSA asymmetric cryptography to encrypt symmetric keys for exchange. However, unlike TR-34, this method does not have the payload signed by the sending party. Also, this RSA wrap technique does not maintain the integrity of the key metadata during transfer by virtue of not including key blocks.

Note

RSA wrap can be used to import or export TDES and AES-128 keys.

1. Call the initialize import command

Call `get-parameters-for-import` to initialize the import process with a key material type of `KEY_CRYPTOGRAM`. `WrappingKeyAlgorithm` can be `RSA_2048` when exchanging TDES keys. `RSA_3072` or `RSA_4096` can be used when exchanging TDES or AES-128 keys. This API will generate a keypair for the purpose of key imports, sign the key using a certificate root and return back both the certificate and certificate root. Ultimately, the key to be exported should be encrypted using this key. Note that these certificates are short lived and are only intended for this purpose.

```
$ 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. Install public certificate on key source system

With many HSMs, you may need to install/load/trust the public certificate(and/or its root) generated in step 1 in order to export keys using it.

3. Export key from source system

Many HSMs and related systems support the ability to export keys using RSA wrap. You'll want to specify the public key from step 1 as the (encryption) cert (WrappingKeyCertificate). If you need the chain of trust, this is contained in the response field WrappingKeyCertificateChain in step #1. When exporting the key from your HSM, you'll want to specify the format to be RSA, Padding Mode = PKCS#1 v2.2 OAEP (with SHA 256 or SHA 512).

4. Call import key

As the last step, you will call the importKey API with a KeyMaterialType of KeyMaterial. You will need the import-token from step 1 and the key-material (wrapped key material) from step 3. You will need to provide the key parameters (such as Key Usage) since RSA wrap does not utilize key blocks.

```
$ 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
```

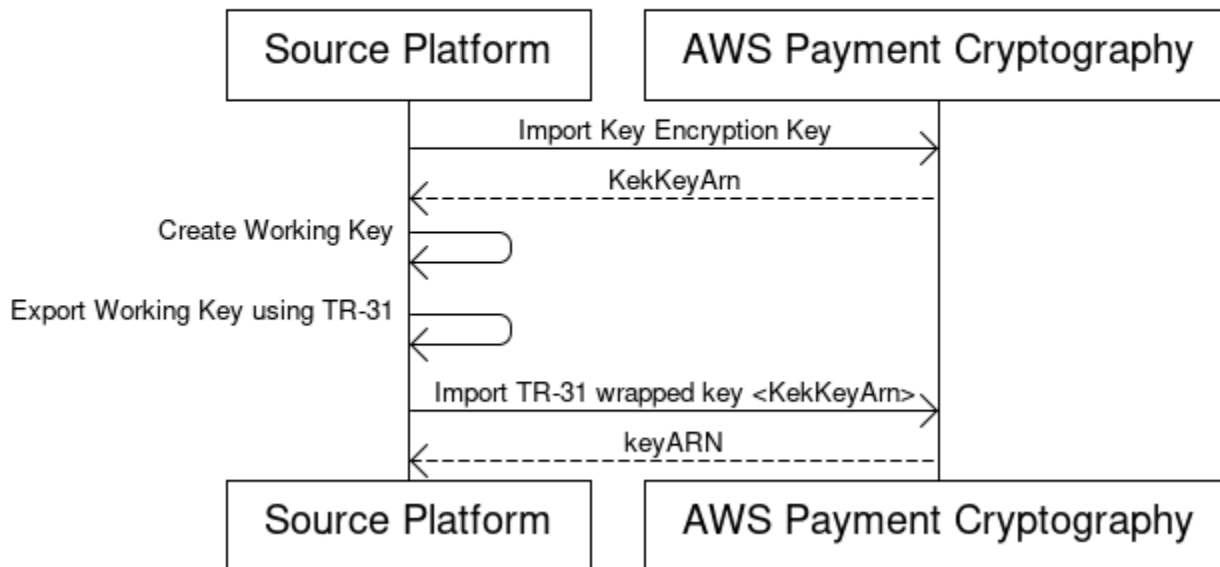
```
{
  "Key": {
    "KeyOrigin": "EXTERNAL",
    "Exportable": true,
    "KeyCheckValue": "DA1ACF",
    "UsageStartTimestamp": 1697643478.92,
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaiifllw2h",
    "CreateTimestamp": 1697643478.92,
    "KeyState": "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. Use imported key for cryptographic operations or subsequent import

If the imported KeyUsage was TR31_K0_KEY_ENCRYPTION_KEY, then this key can be used for subsequent key imports using TR-31. If the key type was any other type (such as TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY), then the key can be directly used for cryptographic operations.

Import symmetric keys using a pre-established key exchange key (TR-31)

Import symmetric keys using a pre-established key exchange key (TR-31)



When partners are exchanging multiple keys (or to support key rotation), it is typical to first exchange an initial key encryption key (KEK) using techniques such as paper key components or in the case of AWS Payment Cryptography using [TR-34](#).

Once a KEK is established, you can use this key to transport subsequent keys (including other KEKs). AWS Payment Cryptography supports this kind of key exchange using ANSI TR-31 which is widely used and widely supported by HSM vendors.

1. Import Key Encryption Key (KEK)

It is assumed that you've already imported your KEK and have the `keyARN` (or `keyAlias`) available to you.

2. Create key on source platform

If the key doesn't already exist, create the key on the source platform. Conversely, you can create the key on AWS Payment Cryptography and use the `export` command instead.

3. Export key from source platform

When exporting, ensure that you specify the export format as TR-31. The source platform will also ask you for the key to be exported and the key encryption key to use.

4. Import into AWS Payment Cryptography

When calling the `importKey` command, `WrappingKeyIdentifier` should be the keyARN (or alias) of your key encryption key and `WrappedKeyBlock` is the output from the source platform.

Example

```
$ aws payment-cryptography import-key \
    --key-material="Tr31KeyBlock={WrappingKeyIdentifier="arn:aws:payment-
cryptography:us-east-2:111122223333:key/ov6icy4ryas4zcza",\
    WrappedKeyBlock="D0112B0AX00E00002E0A3D58252CB67564853373D1EBCC1E23B2ADE7B15E967CC27B85D599"
```

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaiifllw2h",
    "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"
  }
}
```



```
}
}
```

Importing asymmetric (RSA) keys

Importing RSA public keys

AWS Payment Cryptography supports importing public RSA keys in the form of X.509 certificates. In order to import a certificate, you will need to first import its root certificate. All certificates should be unexpired at the time of import. The certificate should be in PEM format and should be base64 encoded.

1. Import into Root Certificate into AWS Payment Cryptography

Example

```
$ aws payment-cryptography import-key \
  --key-material='{"RootCertificatePublicKey":{"KeyAttributes":
{"KeyAlgorithm":"RSA_2048", \
  "KeyClass":"PUBLIC_KEY", "KeyModesOfUse":{"Verify":
true},"KeyUsage":"TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"}, \
  "PublicKeyCertificate":"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSURKVENDQWcyZ0F3SUJBZ01CWkR"
```

```
{
  "Key": {
    "CreateTimestamp": "2023-08-08T18:52:01.023000+00:00",
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
zabouwe3574jysdl",
    "KeyAttributes": {
      "KeyAlgorithm": "RSA_2048",
      "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": "2023-08-08T18:52:01.023000+00:00"
}
}

```

2. Import Public Key Certificate into AWS Payment Cryptography

You can now import a public key. There are two options for importing public keys. TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE can be used if the purpose of the key is to verify signatures (for instance when importing using TR-34). TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION can be used when encrypting data that is meant for use with another system.

Example

```

$ aws payment-cryptography import-key \
  --key-material='{"TrustedCertificatePublicKey":
{"CertificateAuthorityPublicKeyIdentifier":"arn:aws:payment-cryptography:us-
east-2:111122223333:key/zabouwe3574jysdl", \
  "KeyAttributes":
{"KeyAlgorithm":"RSA_2048","KeyClass":"PUBLIC_KEY","KeyModesOfUse":
{"Verify":true},"KeyUsage":"TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"},\
  "PublicKeyCertificate":"LS0tLS1CRUdJTiB..."}}'

```

```

{
  "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"
}
}
```

Export keys

Topics

- [Exporting symmetric keys](#)
- [Exporting asymmetric \(RSA\) keys](#)

Exporting symmetric keys

Important

Examples may require the latest version of the AWS CLI V2. Before getting started, please ensure that you have upgraded to the [latest version](#).

Topics

- [Export keys using asymmetric techniques \(TR-34\)](#)
- [Export keys using asymmetric techniques \(RSA Wrap\)](#)
- [Export symmetric keys using a pre-established key exchange key \(TR-31\)](#)
- [Export DUKPT Initial Keys \(IPEK/IK\)](#)

- [Specifying key block headers upon export](#)

Export keys using asymmetric techniques (TR-34)

Overview: TR-34 utilizes RSA asymmetric cryptography to encrypt symmetric keys for exchange as well as ensuring the source of the data (signing). This ensures both the confidentiality (encryption) and integrity (signature) of the wrapped key. When exporting, AWS Payment Cryptography becomes the key distribution host (KDH) and the target system becomes the key receiving device (KRD).

1. Call the initialize export command

Call `get-parameters-for-export` to initialize the export process. This API will generate a keypair for the purpose of key exports, sign the key and return back the certificate and certificate root. Ultimately, the private key generated by this command used to sign the export payload. In TR-34 terminology, this is known as the KDH signing cert. Note that these certificates are short lived and are only intended for this purpose. The parameter `ParametersValidUntilTimestamp` specifies their duration.

NOTE: All certificates are returned in a base64 encoded format

Example

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

```
{
  "SigningKeyCertificate":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2RENDQXFTZ0F3SUJBZ01RZFazSzNHNEFKT0I4WTNpTmUvY1
  "SigningKeyCertificateChain":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUY0VENDQThZ0F3SUJBZ01SQUt1N2piaHFKZjJPd3FGUWI5c3
  "SigningKeyAlgorithm": "RSA_2048",
  "ExportToken": "export-token-au7pvkbsq4mbup6i",
  "ParametersValidUntilTimestamp": "2023-06-13T15:40:24.036000-07:00"
}
```

2. Import AWS Payment Cryptography certificate into receiving system

Import the certificate chain provided in step 1 into your receiving system as necessary.

3. Generate a key pair, create a public certificate and provide the certificate root to AWS Payment Cryptography

To ensure confidentiality of the transmitted payload, it is encrypted by the sending party (known as the Key Distribution Host or KDH). The receiving party (typically your HSM or your partners' HSM) will want to generate a public key for this purpose and then create a public key certificate (x.509) that can be provided back to AWS Payment Cryptography. AWS Private CA is one option for generating certificates, but there is no restrictions on the certificate authority used.

Once you have the certificate, you'll want to load the root certificate to AWS Payment Cryptography using the `ImportKey` command and `KeyMaterialType` of `ROOT_PUBLIC_KEY_CERTIFICATE` and `KeyUsageType` of `TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`.

The `KeyUsageType` of this certificate is `TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE` because it is the root key and is used to sign the leaf certificate. Leaf certificates for import/export are not imported into AWS Payment Cryptography but are passed inline.

Note

If the root certificate was previously imported, this step can be skipped.

4. Call Export key

As the last step, you will call the `ExportKey` API with a `KeyMaterialType` of `TR34_KEY_BLOCK`. The `certificate-authority-public-key-identifier` will be the `keyARN` of the root CA import in step 3, `WrappingKeyCertificate` will be leaf certificate from step 3 and `export-key-identifier` is the `keyARN` (or alias) to be exported. You will also need to provide the `export-token` from step 1.

Export keys using asymmetric techniques (RSA Wrap)

Overview: AWS Payment Cryptography supports RSA wrap/unwrap for key exchange when TR-34 is not an option available by the counter party. Similar to TR-34, this technique utilizes RSA asymmetric cryptography to encrypt symmetric keys for exchange. However, unlike TR-34, this method does not have the payload signed by the sending party. Also, this RSA wrap technique does not include key blocks which are used to maintain the integrity of key metadata during transport.

Note

RSA wrap can be used to export TDES and AES-128 keys.

1. Generate an RSA key and certificate on receiving system

Create (or identify) an RSA key that will be used for receiving the wrapped key. AWS Payment Cryptography expects keys in X.509 certificate format. Certificate should be signed by a root certificate that is imported (or can be imported) into AWS Payment Cryptography.

2. Install root public certificate on AWS Payment Cryptography

```
$ 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":"LS
```

```
{
  "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. Call export key

Next you want to instruct AWS Payment Cryptography to export your key using your leaf certificate. You will specify the ARN for the previously imported root certificate, the leaf certificate to use for export and the symmetric key to export. The output will be a hex encoded binary wrapped (encrypted) version of your symmetric key.

```
$ 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/zabouwe3574jysd1",
      "WrappingKeyCertificate": "LS0tLS1CRUdJTiBD...",
      "WrappingSpec": "RSA_OAEP_SHA_256"
    }
  }
}

```

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

```

{
  "WrappedKey": {
    "KeyMaterial":
"18874746731E9E1C4562E4116D1C2477063FCB08454D757D81854AEAEE0A52B1F9D303FA29C02DC82AE7785353"
    "WrappedKeyMaterialFormat": "KEY_CRYPTOGRAM"
  }
}

```

4. Import key to receiving system

Many HSMs and related systems support the ability to import keys using RSA unwrap (including AWS Payment Cryptography). In order to do so, specify the public key from step 1 as the (encryption) cert. and the format should be specified as RSA, Padding Mode = PKCS#1 v2.2 OAEP (with SHA 256). The exact terminology may vary by HSM.

Note

AWS Payment Cryptography outputs the wrapped key in hexBinary. You may need to convert the format before importing if your system requires a different binary representation like base64.

Export symmetric keys using a pre-established key exchange key (TR-31)

When partners are exchanging multiple keys (or to support key rotation), it is typical to first exchange an initial key encryption key (KEK) using techniques such as paper key components or in the case of AWS Payment Cryptography using [TR-34](#). Once a KEK is established, you can use this key to transport subsequent keys (including other KEK). AWS Payment Cryptography supports this kind of key exchange using ANSI TR-31 which is widely used and widely supported by HSM vendors.

1. Exchange Key Encryption Key (KEK)

It is assumed that you've already exchange your KEK and have the keyARN (or keyAlias) available to you.

2. Create key on AWS Payment Cryptography

If the key doesn't already exist, create the key. Conversely, you can create the key on the other system and use the [import](#) command instead.

3. Export key from AWS Payment Cryptography

When exporting, the format will be TR-31. When calling the API, you will specify the key to be exported and the wrapping key to be used.

```
$ 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/5rplquuwozodpwp
```



```
{
  "WrappedKey": {
    "KeyCheckValue": "73C263",
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "KeyMaterial":
      "D0144K0AB00E0000A24D3ACF3005F30A6E31D533E07F2E1B17A2A003B338B1E79E5B3AD4FBF7850FACF9A37844",
    "WrappedKeyMaterialFormat": "TR31_KEY_BLOCK"
  }
}
```

4. Import into your system

You or your partner will use the import key implementation on your system to import the key.

Export DUKPT Initial Keys (IPEK/IK)

When using [DUKPT](#), a single Base Derivation Key(BDK) may be generated for a fleet of terminals. Terminals, however, never have access to that original BDK but are each injected with a unique, initial terminal key known as IPEK or Initial Key(IK). Each IPEK is a key derived from the BDK and is intended to be unique per terminal but is derived from the original BDK. The derivation data for this calculation is known as the Key Serial Number (KSN). Per X9.24, for TDES the 10 byte KSN typically consists of 24 bits for the Key Set ID, 19 bits for the terminal ID and 21 bits for the transaction counter. For AES, the 12 byte KSN typically consists of 32 bits for the BDK ID, 32 bits for the derivation identifier(ID) and 32 bits for the transaction counter.

AWS Payment Cryptography provides a mechanism to generate and export these initial keys. Once generated, these keys can be exported using the TR-31, TR-34 and RSA wrap methods. IPEK keys are not persisted and cannot be used for subsequent operations on AWS Payment Cryptography

AWS Payment Cryptography does not enforce the split between the first two parts of the KSN. If you wish to store the derivation identifier along with the BDK, you can use the AWS tags feature for this purpose.

Note

The counter portion of the KSN (32 bits for AES DUKPT) is not used for IPEK/IK derivation. Therefore, an input of 12345678901234560001 and 12345678901234569999 will output the same IPEK.

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

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

Specifying key block headers upon export

AWS Payment Cryptography supports the ability to modify or append key block information when exporting in ASC TR-31 or TR-34 formats. The below table describes the TR-31 key block format and which elements can be modified upon export.

Key Block Attribute	Purpose	Modify or Append Upon Export?	Notes
Version ID	Represents the method used to protect the underlying key material. The standard defines version A and version C (key variant), version B (derivation using TDES) and version D (key derivation using AES).	Not modifiable	AWS Payment Cryptography will use version B when the wrapping key is TDES and version D when the wrapping key is AES. Version A and Version C are deprecated and only supported for import operations.

Key Block Attribute	Purpose	Modify or Append Upon Export?	Notes
Key Block Length	Represents the length of the remaining message	Not modifiable	This value is automatically calculated by the service. Note that the service may employ key padding as required by the specification, so the length may not appear correct prior to decrypting the payload.
Key Usage	Represents the permitted purposes of the key such as C0 (Card Verification) or B0 (Base Derivation Key)	Not modifiable	
Algorithm	Represents the algorithm of the underlying key. The standard supports multiple key types such as T (TDES), A(AES) and E(ECC). AWS Payment Cryptography currently supports values of T,H and A.	Not modifiable	This is exported from AWS Payment Cryptography as-is.

Key Block Attribute	Purpose	Modify or Append Upon Export?	Notes
Key Usage	Represents the types of operations it can be used for. Examples include Generate and Verify (C) and Encrypt/Decrypt/Wrap/Unwrap(B)	Modify*	
Key Version	Represents the version number of the key if the key is replaced/rotated. This is a numeric value and defaults to 00 if not specified.	Append	
Key Exportability	Represents whether the key can be exported. N means No Exportability, E means export according to X9.24 (key blocks), S means export under key block or non-key block formats.	Modify*	

Key Block Attribute	Purpose	Modify or Append Upon Export?	Notes
Optional Key Blocks	Append	Optional key blocks are a series of name/value pairs that can be cryptically bound to the key. A common example is KeySetID for DUKPT keys. The actual format includes the number of optional blocks, the length of each one and a padding block (PB), but AWS Payment Cryptography will automatically calculate these based on name/value pair input.	

**When modifying, the new value must be more restrictive than the current value within AWS Payment Cryptography. For instance, if current key mode of use is `Generate=True,Verify=True`, when exporting, you can change it to `Generate=True,Verify=False`. Similarly if the key is already set to not exportable, you can not change it to exportable.*

When exporting keys, AWS Payment Cryptography automatically applies the current values for the key being exported without modification. However, in some cases, you may wish to modify or append those values before sending to the receiving system even if that is permitted by within AWS Payment Cryptography. One example is that when exporting a key to a payment terminal, it is typical to restrict its exportability to `NotExportable` as terminals are typically only meant to import keys and thus should not be subsequently exporting keys.

Another example is when you want to pass associated key metadata to the receiving system. Prior to TR-31, common practice was to create a custom payload to send such information, but using TR-31 you can cryptographically bind it to the key in a specific format. Key Version can be

set using the `KeyVersion` field. TR-31/X9.143 specifies certain common headers but there is no restriction on using others as long as it fits within the AWS Payment Cryptography parameters and the receiving system is able to accept it. For more information on specifying key blocks during export, please consult the [Key Block Headers](#) in the API Guide.

In this example, we are exporting a BDK key (for instance to a KIF). We are specifying the key version as 02, setting the `KeyExportability` to `NON_EXPORTABLE` and providing an optional value for the `KeySetID` of 00ABCDEFAB meaning it is a TDES key (00) and the initial key is ABCDEFABCD. As key modes of use is not specified, this key will inherit the mode of use from `arn:aws:payment-cryptography:us-east-2:111122223333:key/5rplquuwozodpwsp` which is `DeriveKey = true`

Note

Even though exportability is set to Not Exportable in this example, the [KIF](#) will still be able to derive keys such as a [IPEK/IK](#) used in DUKPT and these derived keys will be exportable in order to install on the devices. This is specifically permitted by the standards.

```
$ aws payment-cryptography --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":
"D0128B0TX02N0100BI1000ABCDEFABCD1A2C0EADE244321640E94FE3A3C9D33800D47CE64238D9327DDBFE25B9023
    "KeyCheckValue": "A4C9B3",
    "KeyCheckValueAlgorithm": "ANSI_X9_24"
  }
}
```

Exporting asymmetric (RSA) keys

Call `get-public-key-certificate` to export a public key in certificate form. This API will export the certificate as well as its root certificate encoded in base64 format.

NOTE: This API is not idempotent - subsequent calls may result in different certificates even though the underlying key is the same.

Example

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

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

Using aliases

An *alias* is a friendly name for an AWS Payment Cryptography key. For example, an alias lets you refer to a key as `alias/test-key` instead of `arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h`.

You can use an alias to identify a key in most key management (control plane) operations, and in [cryptographic \(dataplane\) operations](#).

You can also allow and deny access to AWS Payment Cryptography key based on their aliases without editing policies or managing grants. This feature is part of the service's support for [attribute-based access control](#) (ABAC).

Much of the power of aliases comes from your ability to change the key associated with an alias at any time. Aliases can make your code easier to write and maintain. For example, suppose you use an alias to refer to a particular AWS Payment Cryptography key and you want to change the AWS Payment Cryptography key. In that case, just associate the alias with a different key. You don't need to change your code or application configuration.

Aliases also make it easier to reuse the same code in different AWS Regions. Create aliases with the same name in multiple Regions and associate each alias with an AWS Payment Cryptography key in its Region. When the code runs in each Region, the alias refers to the associated AWS Payment Cryptography key in that Region.

You can create an alias for an AWS Payment Cryptography key by using the `CreateAlias` API.

The AWS Payment Cryptography API provides full control of aliases in each account and Region. The API includes operations to create an alias (`CreateAlias`), view alias names and the linked keyARN (`list-aliases`), change the AWS Payment Cryptography key associated with an alias (`update-alias`), and delete an alias (`delete-alias`).

Topics

- [About aliases](#)
- [Using aliases in your applications](#)
- [Related APIs](#)

About aliases

Learn how aliases work in AWS Payment Cryptography.

An alias is an independent AWS resource

An alias is not a property of an AWS Payment Cryptography key. The actions that you take on the alias don't affect its associated key. You can create an alias for an AWS Payment Cryptography key and then update the alias so it's associated with a different AWS Payment Cryptography key. You can even delete the alias without any effect on the associated AWS Payment Cryptography key. If you delete a AWS Payment Cryptography key, all aliases associated with that key will become unassigned.

If you specify an alias as the resource in an IAM policy, the policy refers to the alias, not to the associated AWS Payment Cryptography key.

Each alias has a friendly name

When you create an alias, you specify the alias name prefixed by `alias/`. For instance `alias/test_1234`

Each alias is associated with one AWS Payment Cryptography key at a time

The alias and its AWS Payment Cryptography key must be in the same account and Region.

An AWS Payment Cryptography key can be associated with more than one alias concurrently, but each alias can only be mapped to a single key

For example, this `list-aliases` output shows that the `alias/sampleAlias1` alias is associated with exactly one target AWS Payment Cryptography key, which is represented by the `KeyArn` property.

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

```
{
  "Aliases": [
    {
      "AliasName": "alias/sampleAlias1",
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaif1lw2h"
    }
  ]
}
```

Multiple aliases can be associated with the same AWS Payment Cryptography key

For example, you can associate the `alias/sampleAlias1`; and `alias/sampleAlias2` aliases with the same key.

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

```
{
  "Aliases": [
    {
      "AliasName": "alias/sampleAlias1",
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaif1lw2h"
    },
    {
      "AliasName": "alias/sampleAlias2",
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaif1lw2h"
    }
  ]
}
```

```
}
```

An alias must be unique for a given account and Region

For example, you can have only one `alias/sampleAlias1` alias in each account and Region. Aliases are case-sensitive, but we recommend against using aliases that only differ in capitalization as they can be prone to error. You cannot change an alias name. However, you can delete the alias and create a new alias with the desired name.

You can create an alias with the same name in different Regions

For example, you can have alias `alias/sampleAlias2` in US East (N. Virginia) and alias `alias/sampleAlias2` in US West (Oregon). Each alias would be associated with an AWS Payment Cryptography key in its Region. If your code refers to an alias name like `alias/finance-key`, you can run it in multiple Regions. In each Region, it uses a different `alias/sampleAlias2`. For details, see [Using aliases in your applications](#).

You can change the AWS Payment Cryptography key associated with an alias

You can use the `UpdateAlias` operation to associate an alias with a different AWS Payment Cryptography key. For example, if the `alias/sampleAlias2` alias is associated with the `arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaiif1lw2h` AWS Payment Cryptography key, you can update it so it is associated with the `arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi` key.

Warning

AWS Payment Cryptography doesn't validate that the old and new keys have all the same attributes such as key usage. Updating with a different key type may result in problems in your application.

Some keys don't have aliases

An alias is an optional feature and not all keys will have aliases unless you choose to operate your environment in this way. Keys can be associated with Aliases using the `create-alias` command. Also, you can use the `update-alias` operation to change the AWS Payment Cryptography key associated with an alias and the `delete-alias` operation to delete an alias. As a result, some AWS Payment Cryptography keys might have several aliases, and some might have none.

Mapping a key to an alias

You can map a key (represented by an ARN) to one or more aliases using the `create-alias` command. This command is not idempotent - to update an alias, use the `update-alias` command.

```
$ aws payment-cryptography create-alias --alias-name alias/sampleAlias1 \  
    --key-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/  
    kwapwa6qaiif1lw2h
```

```
{  
  "Alias": {  
    "AliasName": "alias/alias/sampleAlias1",  
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/  
    kwapwa6qaiif1lw2h"  
  }  
}
```

Using aliases in your applications

You can use an alias to represent an AWS Payment Cryptography key in your application code. The `key-identifier` parameter in AWS Payment Cryptography [data operations](#) as well as other operations like List Keys accepts an alias name or alias ARN.

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

When using an alias ARN, remember that the alias mapping to an AWS Payment Cryptography key is defined in the account that owns the AWS Payment Cryptography key and might differ in each Region.

One of the most powerful uses of aliases is in applications that run in multiple AWS Regions.

You could create a different version of your application in each Region or use a dictionary, configuration or switch statement to select the right AWS Payment Cryptography key for each Region. But it might be easier to create an alias with the same alias name in each Region. Remember that the alias name is case-sensitive.

Related APIs

[Tags](#)

Tags are key and value pairs that act as metadata for organizing your AWS Payment Cryptography keys. They can be used to flexibly identify keys or group one or more keys together.

Get keys

An AWS Payment Cryptography key represents a single unit of cryptographic material and can only be used for cryptographic operations for this service. The `GetKeys` API takes a `KeyIdentifier` as input and returns the immutable and mutable attributes of the key but does not contain any cryptographic material.

Example

```
$ aws payment-cryptography get-key --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h
```

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h",
    "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"
  }
}
```

Get the public key/certificate associated with a key pair

Get Public Key/Certificate returns the public key indicated by the `KeyArn`. This can be the public key portion of a key pair generated on AWS Payment Cryptography or a previously imported public key. The most common use case is to provide the public key to an outside service that will encrypt data. That data can then be passed to an application leveraging AWS Payment Cryptography and the data can be decrypted using the private key secured within AWS Payment Cryptography.

The service returns public keys as a public certificate. The API result contains the CA and the public key certificate. Both data elements are base64 encoded.

Note

The public certificate returned is intended to be short lived and is not intended to be idempotent. You may receive a different certificate on each API call even the public key itself is unchanged.

Example

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

```
{
  "KeyCertificate":
  "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2VENDQXFXZ0F3SUJBZ01SQUo10Wd2VkpDd3d1Y1dMNldYZEpYY
  "KeyCertificateChain":
  "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUY0VENDQThZ0F3SUJBZ01SQUt1N2piaHFKZjJPd3FGUWI5c3VuO
}
```

Tagging keys

In AWS Payment Cryptography, you can add tags to a AWS Payment Cryptography key when you [create a key](#), and tag or untag existing keys unless they are pending deletion. Tags are optional, but they can be very useful.

For general information about tags, including best practices, tagging strategies, and the format and syntax of tags, see [Tagging AWS resources](#) in the *Amazon Web Services General Reference*.

Topics

- [About tags in AWS Payment Cryptography](#)
- [Viewing key tags in the console](#)
- [Managing key tags with API operations](#)
- [Controlling access to tags](#)
- [Using tags to control access to keys](#)

About tags in AWS Payment Cryptography

A *tag* is an optional metadata label that you can assign (or AWS can assign) to an AWS resource. Each tag consists of a *tag key* and a *tag value*, both of which are case-sensitive strings. The tag value can be an empty (null) string. Each tag on a resource must have a different tag key, but you can add the same tag to multiple AWS resources. Each resource can have up to 50 user-created tags.

Do not include confidential or sensitive information in the tag key or tag value. Tags are accessible to many AWS services, including billing.

In AWS Payment Cryptography, you can add tags to a key when you [create the key](#), and tag or untag existing keys unless they are pending deletion. You cannot tag aliases. Tags are optional, but they can be very useful.

For example, you can add a "Project"="Alpha" tag to all AWS Payment Cryptography keys and Amazon S3 buckets that you use for the Alpha project. Another example is to add "BIN"="20130622" tag to all keys associated to a specific bank identification number(BIN).

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

For general information about tags, including the format and syntax, see [Tagging AWS resources](#) in the *Amazon Web Services General Reference*.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to an AWS Payment Cryptography keys and an Amazon Elastic Block Store (Amazon EBS) volume or AWS Secrets Manager secret. You can also use tags to identify keys for automation.
- Track your AWS costs. When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. You can use this feature to track AWS Payment Cryptography costs for a project, application, or cost center.

For more information about using tags for cost allocation, see [Using Cost Allocation Tags](#) in the *AWS Billing User Guide*. For information about the rules for tag keys and tag values, see [User-Defined Tag Restrictions](#) in the *AWS Billing User Guide*.

- Control access to your AWS resources. Allowing and denying access to keys based on their tags is part of AWS Payment Cryptography support for attribute-based access control (ABAC). For

information about controlling access to AWS Payment Cryptography based on their tags, see [Authorization based on AWS Payment Cryptography tags](#). For more general information about using tags to control access to AWS resources, see [Controlling Access to AWS Resources Using Resource Tags](#) in the *IAM User Guide*.

AWS Payment Cryptography writes an entry to your AWS CloudTrail log when you use the `TagResource`, `UntagResource`, or `ListTagsForResource` operations.

Viewing key tags in the console

To view tags in the console, you need tagging permission on the key from an IAM policy that includes the key. You need these permissions in addition to the permissions to view keys in the console.

Managing key tags with API operations

You can use the [AWS Payment Cryptography API](#) to add, delete, and list tags for the keys that you manage. These examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language. You cannot tag AWS managed keys.

To add, edit, view, and delete tags for a key, you must have the required permissions. For details, see [Controlling access to tags](#).

Topics

- [CreateKey: Add tags to a new key](#)
- [TagResource: Add or change tags for a key](#)
- [ListResourceTags: Get the tags for a key](#)
- [UntagResource: Delete tags from a key](#)

CreateKey: Add tags to a new key

You can add tags when you create a key. To specify the tags, use the `Tags` parameter of the [CreateKey](#) operation.

To add tags when creating a key, the caller must have `payment-cryptography:TagResource` permission in an IAM policy. At a minimum, the permission must cover all keys in the account and Region. For details, see [Controlling access to tags](#).

The value of the `Tags` parameter of `CreateKey` is a collection of case-sensitive tag key and tag value pairs. Each tag on a key must have a different tag name. The tag value can be a null or empty string.

For example, the following AWS CLI command creates a symmetric encryption key with a `Project:Alpha` tag. When specifying more than one key-value pair, use a space to separate each pair.

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDDES_2KEY, \
  KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY, \
  KeyModesOfUse='{Generate=true,Verify=true}' \
  --tags '[{"Key":"Project","Value":"Alpha"}, {"Key":"BIN","Value":"123456"}]'
```

When this command is successful, it returns a `Key` object with information about the new key. However, the `Key` does not include tags. To get the tags, use the [ListResourceTags](#) operation.

TagResource: Add or change tags for a key

The [TagResource](#) operation adds one or more tags to a key. You cannot use this operation to add or edit tags in a different AWS account.

To add a tag, specify a new tag key and a tag value. To edit a tag, specify an existing tag key and a new tag value. Each tag on a key must have a different tag key. The tag value can be a null or empty string.

For example, the following command adds `UseCase` and `BIN` tags to an example key.

```
$ aws payment-cryptography tag-resource --resource-arn arn:aws:payment-
cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h --tags
'[{"Key":"UseCase","Value":"Acquiring"}, {"Key":"BIN","Value":"123456"}]'
```

When this command is successful, it does not return any output. To view the tags on a key, use the [ListResourceTags](#) operation.

You can also use `TagResource` to change the tag value of an existing tag. To replace a tag value, specify the same tag key with a different value. Tags not listed in a modify command are not changed or removed.

For example, this command changes the value of the `Project` tag from `Alpha` to `Noe`.

The command will return `http/200` with no content. To see your changes, use `ListTagsForResource`

```
$ aws payment-cryptography tag-resource --resource-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h \  
    --tags '[{"Key":"Project","Value":"Noe"}]'
```

ListResourceTags: Get the tags for a key

The [ListResourceTags](#) operation gets the tags for a key. The `ResourceArn` (`keyArn` or `keyAlias`) parameter is required. You cannot use this operation to view the tags on keys in a different AWS account.

For example, the following command gets the tags for an example key.

```
$ aws payment-cryptography list-tags-for-resource --resource-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaif1lw2h  
  
{  
  "Tags": [  
    {  
      "Key": "BIN",  
      "Value": "20151120"  
    },  
    {  
      "Key": "Project",  
      "Value": "Production"  
    }  
  ]  
}
```

UntagResource: Delete tags from a key

The [UntagResource](#) operation deletes tags from a key. To identify the tags to delete, specify the tag keys. You cannot use this operation to delete tags from keys a different AWS account.

When it succeeds, the `UntagResource` operation doesn't return any output. Also, if the specified tag key isn't found on the key, it doesn't throw an exception or return a response. To confirm that the operation worked, use the [ListResourceTags](#) operation.

For example, this command deletes the **Purpose** tag and its value from the specified key.

```
$ aws payment-cryptography untag-resource \  
    --resource-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/  
    kwapwa6qaif1lw2h --tag-keys Project
```

Controlling access to tags

To add, view, and delete tags by using the API, principals need tagging permissions in IAM policies.

You can also limit these permissions by using AWS global condition keys for tags. In AWS Payment Cryptography, these conditions can control access to tagging operations, such as [TagResource](#) and [UntagResource](#).

For example policies and more information, see [Controlling Access Based on Tag Keys](#) in the *IAM User Guide*.

Permissions to create and manage tags work as follows.

payment-cryptography:TagResource

Allows principals to add or edit tags. To add tags while creating a key, the principal must have permission in an IAM policy that isn't restricted to particular keys.

payment-cryptography:ListTagsForResource

Allows principals to view tags on keys.

payment-cryptography:UntagResource

Allows principals to delete tags from keys.

Tag permissions in policies

You can provide tagging permissions in a key policy or IAM policy. For example, the following example key policy gives select users tagging permission on the key. It gives all users who can assume the example Administrator or Developer roles permission to view tags.

```
{  
  "Version": "2012-10-17",  
  "Id": "example-key-policy",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",
```

```

    "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
    "Action": "payment-cryptography:*",
    "Resource": "*"
  },
  {
    "Sid": "Allow all tagging permissions",
    "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:ListResourceTags",
    "Resource": "*"
  }
]
}

```

To give principals tagging permission on multiple keys, you can use an IAM policy. For this policy to be effective, the key policy for each key must allow the account to use IAM policies to control access to the key.

For example, the following IAM policy allows the principals to create keys. It also allows them to create and manage tags on all keys in the specified account. This combination allows the principals to use the tags parameter of the [CreateKey](#) operation to add tags to a key while they are creating it.

```

{
  "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/*"
}
]
```

Limiting tag permissions

You can limit tagging permissions by using policy conditions. The following policy conditions can be applied to the `payment-cryptography:TagResource` and `payment-cryptography:UntagResource` permissions. For example, you can use the `aws:RequestTag/tag-key` condition to allow a principal to add only particular tags, or prevent a principal from adding tags with particular tag keys.

- [aws:RequestTag](#)
- [aws:ResourceTag/tag-key](#) (IAM policies only)
- [aws:TagKeys](#)

As a best practice when you use tags to control access to keys, use the `aws:RequestTag/tag-key` or `aws:TagKeys` condition key to determine which tags (or tag keys) are allowed.

For example, the following IAM policy is similar to the previous one. However, this policy allows the principals to create tags (`TagResource`) and delete tags `UntagResource` only for tags with a `Project` tag key.

Because `TagResource` and `UntagResource` requests can include multiple tags, you must specify a `ForAllValues` or `ForAnyValue` set operator with the [aws:TagKeys](#) condition. The

`ForAnyValue` operator requires that at least one of the tag keys in the request matches one of the tag keys in the policy. The `ForAllValues` operator requires that all of the tag keys in the request match one of the tag keys in the policy. The `ForAllValues` operator also returns `true` if there are no tags in the request, but `TagResource` and `UntagResource` fail when no tags are specified. For details about the set operators, see [Use multiple keys and values](#) in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyCreateKey",
      "Effect": "Allow",
      "Action": "payment-cryptography:CreateKey",
      "Resource": "*"
    },
    {
      "Sid": "IAMPolicyViewAllTags",
      "Effect": "Allow",
      "Action": "payment-cryptography:ListResourceTags",
      "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"}
      }
    }
  ]
}
```

Using tags to control access to keys

You can control access to AWS Payment Cryptography based on the tags on the key. For example, you can write an IAM policy that allows principals to enable and disable only the keys that have a particular tag. Or you can use an IAM policy to prevent principals from using keys in cryptographic operations unless the key has a particular tag.

This feature is part of AWS Payment Cryptography support for attribute-based access control (ABAC). For information about using tags to control access to AWS resources, see [What is ABAC for AWS?](#) and [Controlling Access to AWS Resources Using Resource Tags](#) in the *IAM User Guide*.

 **Note**

AWS Payment Cryptography supports the [aws:ResourceTag/tag-key](#) global condition context key, which lets you control access to keys based on the tags on the key. Because multiple keys can have the same tag, this feature lets you apply the permission to a select set of keys. You can also easily change the keys in the set by changing their tags.

In AWS Payment Cryptography, the `aws:ResourceTag/tag-key` condition key is supported only in IAM policies. It isn't supported in key policies, which apply only to one key, or on operations that don't use a particular key, such as the [ListKeys](#) or [ListAliases](#) operations.

Controlling access with tags provides a simple, scalable, and flexible way to manage permissions. However, if not properly designed and managed, it can allow or deny access to your keys inadvertently. If you are using tags to control access, consider the following practices.

- Use tags to reinforce the best practice of [least privileged access](#). Give IAM principals only the permissions they need on only the keys they must use or manage. For example, use tags to label the keys used for a project. Then give the project team permission to use only keys with the project tag.
- Be cautious about giving principals the `payment-cryptography:TagResource` and `payment-cryptography:UntagResource` permissions that let them add, edit, and delete tags. When you use tags to control access to keys, changing a tag can give principals permission to use keys that they didn't otherwise have permission to use. It can also deny access to keys that other principals require to do their jobs. Key administrators who don't have permission to change key policies or create grants can control access to keys if they have permission to manage tags.

Whenever possible, use a policy condition, such as `aws:RequestTag/tag-key` or `aws:TagKeys` to [limit a principal's tagging permissions](#) to particular tags or tag patterns on particular keys.

- Review the principals in your AWS account that currently have tagging and untagging permissions and adjust them, if necessary. IAM policies might allow tag and untag permissions

on all keys. For example, the *Admin* managed policy allows principals to tag, untag, and list tags on all keys.

- Before setting a policy that depends on a tag, review the tags on the keys in your AWS account. Make sure that your policy applies only to the tags you intend to include. Use [CloudTrail logs](#) and CloudWatch alarms to alert you to tag changes that might affect access to your keys.
- The tag-based policy conditions use pattern matching; they aren't tied to a particular instance of a tag. A policy that uses tag-based condition keys affects all new and existing tags that match the pattern. If you delete and recreate a tag that matches a policy condition, the condition applies to the new tag, just as it did to the old one.

For example, consider the following IAM policy. It allows the principals to call the [Decrypt](#) operations only on keys in your account that are the US East (N. Virginia) Region and have a "Project"="Alpha" tag. You might attach this policy to roles in the example Alpha project.

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

The following example IAM policy allows the principals to use any key in the account for certain cryptographic operations. But it prohibits the principals from using these cryptographic operations on keys with a "Type"="Reserved" tag or no "Type" tag.

```
{
  "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"
      }
    }
  }
]
}

```

Understanding key attributes for AWS Payment Cryptography key

A tenet of proper key management is that keys are appropriately scoped and can only be used for permitted operations. As such, certain keys can only be created with certain key modes of use. Whenever possible, this aligns with the available modes of use as defined by [TR-31](#).

Although AWS Payment Cryptography will prevent you from creating invalid keys, valid combinations are provided here for your convenience.

Symmetric Keys

- TR31_B0_BASE_DERIVATION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true },{ NoRestrictions = true }
- TR31_C0_CARD_VERIFICATION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Generate = true },{ Verify = true },{ Generate = true, Verify= true },{ NoRestrictions = true }
- TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } , { NoRestrictions = true }
- TR31_E0_EMV_MKEY_APP_CRYPTGRAMS
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true }, { NoRestrictions = true }
- TR31_E1_EMV_MKEY_CONFIDENTIALITY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true }, { NoRestrictions = true }
- TR31_E2_EMV_MKEY_INTEGRITY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true }, { NoRestrictions = true }

- TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true }, { NoRestrictions = true }
- TR31_E5_EMV_MKEY_CARD_PERSONALIZATION
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true }, { NoRestrictions = true }
- TR31_E6_EMV_MKEY_OTHER
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { DeriveKey = true }, { NoRestrictions = true }
- TR31_K0_KEY_ENCRYPTION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } , { NoRestrictions = true }
- TR31_K1_KEY_BLOCK_PROTECTION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { 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
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY
 - **Allowed combination of key modes of use:** { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_M3_ISO_9797_3_MAC_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY
 - **Allowed combination of key modes of use:** { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_M6_ISO_9797_5_CMAC_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_M7_HMAC_KEY

- **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
- **Allowed combination of key modes of use:** { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_P0_PIN_ENCRYPTION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } ,{ Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true } , { NoRestrictions = true }
- TR31_V1_IBM3624_PIN_VERIFICATION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }
- TR31_V2_VISA_PIN_VERIFICATION_KEY
 - **Allowed Key Algorithms:** TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
 - **Allowed combination of key modes of use:** { Generate = true } ,{ Verify = true } ,{ Generate = true, Verify= true } ,{ NoRestrictions = true }

Asymmetric Keys

- TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION
 - **Allowed Key Algorithms:** RSA_2048 ,RSA_3072 ,RSA_4096
 - **Allowed combination of key modes of use:** { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } ,{ Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true }
 - **NOTE:** { Encrypt = true, Wrap = true } is the only valid option when importing a public key that is intended for encrypting data or wrapping a key
- TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE
 - **Allowed Key Algorithms:** RSA_2048 ,RSA_3072 ,RSA_4096
 - **Allowed combination of key modes of use:** { Sign = true } ,{ Verify = true }
 - **NOTE:** { Verify = true } is the only valid option when importing a key meant for signing, such as root certificate, intermediate certificate or signing certificates for TR-34.

Data operations

After you have established an AWS Payment Cryptography key, it can be used to perform cryptographic operations. Different operations perform different types of activity ranging from encryption, hashing as well as domain specific algorithms such as CVV2 generation.

Encrypted data cannot be decrypted without the matching decryption key (the symmetric key or private key depending on the encryption type). Hashing and domain specific algorithms similarly cannot be verified without the symmetric key or public key.

For information on valid key types for specific operations please see [Valid keys for cryptographic operations](#)

Note

We recommend using test data when in a non-production environment. Using production keys and data (PAN, BDK ID, etc.) in a non-production environment may impact your compliance scope such as for PCI DSS and PCI P2PE.

Topics

- [Encrypt, Decrypt and Re-encrypt data](#)
- [Generate and verify card data](#)
- [Generate, translate and verify PIN data](#)
- [Verify auth request \(ARQC\) cryptogram](#)
- [Generate and verify MAC](#)
- [Valid keys for cryptographic operations](#)

Encrypt, Decrypt and Re-encrypt data

Encryption and Decryption methods can be used to encrypt or decrypt data using a variety of symmetric and asymmetric techniques including TDES, AES and RSA. These methods also support keys derived using [DUKPT](#) and [EMV](#) techniques. For use cases where you wish to secure data under a new key without exposing the underlying data, the ReEncrypt command can also be used.

Note

When using the encrypt/decrypt functions, all inputs are assumed to be in hexBinary - for instance a value of 1 will be input as 31 (hex) and a lower case t is represented as 74 (hex). All outputs are in hexBinary as well.

For details on all available options, please consult the API Guide for [Encrypt](#), [Decrypt](#), and [Re-Encrypt](#).

Topics

- [Encrypt data](#)
- [Decrypt data](#)

Encrypt data

The `Encrypt Data` API is used to encrypt data using symmetric and asymmetric data encryption keys as well as [DUKPT](#) and [EMV](#) derived keys. Various algorithms and variations are supported including TDES, RSA and AES.

The primary inputs are the encryption key used to encrypt the data, the plaintext data in hexBinary format to be encrypted and encryption attributes such as initialization vector and mode for block ciphers such as TDES. The plaintext data needs to be in multiples of 8 bytes for TDES, 16 bytes for AES and the length of the key in the case of RSA. Symmetric key inputs (TDES, AES, DUKPT, EMV) should be padded in cases where the input data does not meet these requirements. The following table shows the maximum length of plaintext for each type of key and the padding type that you define in `EncryptionAttributes` for RSA keys.

Padding type	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

Padding type	RSA_2048	RSA_3072	RSA_4096
None	488	744	1000

The primary outputs include the encrypted data as ciphertext in hexBinary format and the checksum value for the encryption key. For details on all available options, please consult the API Guide for [Encrypt](#).

Examples

- [Encrypt data using AES symmetric key](#)
- [Encrypt data using DUKPT key](#)
- [Encrypt data using EMV-derived symmetric key](#)
- [Encrypt data using an RSA key](#)

Encrypt data using AES symmetric key

Note

All examples assume the relevant key already exists. Keys can be created using the [CreateKey](#) operation or imported using the [ImportKey](#) operation.

Example

In this example, we will encrypt plaintext data using a symmetric key which has been created using the [CreateKey](#) Operation or imported using the [ImportKey](#) Operation. For this operation, the key must have KeyModesOfUse set to Encrypt and KeyUsage set to TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY. Please see [Keys for Cryptographic Operations](#) for more options.

```
$ 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"
}
```

Encrypt data using DUKPT key

Example

In this example, we will encrypt plaintext data using a [DUKPT](#) key. AWS Payment Cryptography supports TDES and AES DUKPT keys. For this operation, the key must have `KeyModesOfUse` set to `DeriveKey` and `KeyUsage` set to `TR31_B0_BASE_DERIVATION_KEY`. Please see [Keys for Cryptographic Operations](#) for more options.

```
$ 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"
}
```

Encrypt data using EMV-derived symmetric key

Example

In this example, we will encrypt clear text data using an EMV-derived symmetric key which has already been created. You might use a command such as this to send data to an EMV card.

For this operation, the key must have `KeyModesOfUse` set to `Derive` and `KeyUsage` set to `TR31_E1_EMV_MKEY_CONFIDENTIALITY` or `TR31_E6_EMV_MKEY_OTHER`. Please see [Keys for Cryptographic Operations](#) for more details.

```
$ 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"
}
```

Encrypt data using an RSA key

Example

In this example, we will encrypt plaintext data using an [RSA public key](#) which has been imported using the [ImportKey](#) operation. For this operation, the key must have `KeyModesOfUse` set to `Encrypt` and `KeyUsage` set to `TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION`. Please see [Keys for Cryptographic Operations](#) for more options.

For PKCS #7 or other padding schemes not currently supported, please apply prior to calling the service and select no padding by omitting the padding indicator `'Asymmetric={}'`

```
$ 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"
}
```

Decrypt data

The Decrypt Data API is used to decrypt data using symmetric and asymmetric data encryption keys as well as [DUKPT](#) and [EMV](#) derived keys. Various algorithms and variations are supported including TDES, RSA and AES.

The primary inputs are the decryption key used to decrypt the data, the ciphertext data in hexBinary format to be decrypted and decryption attributes such as initialization vector, mode as block ciphers etc. The primary outputs include the decrypted data as plaintext in hexBinary format and the checksum value for the decryption key. For details on all available options, please consult the API Guide for [Decrypt](#).

Examples

- [Decrypt data using AES symmetric key](#)
- [Decrypt data using DUKPT key](#)
- [Decrypt data using EMV-derived symmetric key](#)
- [Decrypt data using an RSA key](#)

Decrypt data using AES symmetric key

Example

In this example, we will decrypt ciphertext data using a symmetric key. This example shows an AES key but TDES_2KEY and TDES_3KEY are also supported. For this

operation, the key must have `KeyModesOfUse` set to `Decrypt` and `KeyUsage` set to `TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY`. Please see [Keys for Cryptographic Operations](#) for more options.

```
$ 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": "31323334313233343132333431323334"
}
```

Decrypt data using DUKPT key

Note

Using `decrypt-data` with DUKPT for P2PE transactions may return credit card PAN and other cardholder data to your application that will need to be accounted for when determining its PCI DSS scope.

Example

In this example, we will decrypt ciphertext data using a [DUKPT](#) key which has been created using the [CreateKey](#) Operation or imported using the [ImportKey](#) Operation. For this operation, the key must have `KeyModesOfUse` set to `DeriveKey` and `KeyUsage` set to `TR31_B0_BASE_DERIVATION_KEY`. Please see [Keys for Cryptographic Operations](#) for more options. When you use DUKPT, for TDES algorithm, the ciphertext data length must be a multiple of 16 bytes. For AES algorithm, the ciphertext data length must be a multiple of 32 bytes.

```
$ aws payment-cryptography-data decrypt-data --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
```

```
--cipher-text 33612AB9D6929C3A828EB6030082B2BD --decryption-attributes
'Dukpt={KeySerialNumber=FFFF9876543210E00001}'
```

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

Decrypt data using EMV-derived symmetric key

Example

In this example, we will decrypt ciphertext data using an EMV-derived symmetric key which has been created using the [CreateKey](#) operation or imported using the [ImportKey](#) operation. For this operation, the key must have `KeyModesOfUse` set to `Derive` and `KeyUsage` set to `TR31_E1_EMV_MKEY_CONFIDENTIALITY` or `TR31_E6_EMV_MKEY_OTHER`. Please see [Keys for Cryptographic Operations](#) for more details.

```
$ 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"
}
```

Decrypt data using an RSA key

Example

In this example, we will decrypt ciphertext data using an [RSA key pair](#) which has been created using the [CreateKey](#) operation. For this operation, the key must have `KeyModesOfUse` set to enable Decrypt and `KeyUsage` set to `TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION`. Please see [Keys for Cryptographic Operations](#) for more options.

For PKCS #7 or other padding schemes not currently supported, please select no padding by omitting the padding indicator `'Asymmetric={}'` and remove padding subsequent to calling the service.

```
$ 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=0AEP_SHA256}'
```

```
{  
  "KeyArn": "arn:aws:payment-cryptography:us-  
east-1:111122223333:key/5dza7xqd6soanjtb",  
  "KeyCheckValue": "FF9DE9CE",  
  "PlainText": "31323334313233343132333431323334"  
}
```

Generate and verify card data

Generate and verify card data incorporates data derived from card data, for instance CVV, CVV2, CVC and DCVV.

Topics

- [Generate card data](#)
- [Verify card data](#)

Generate card data

The Generate Card Data API is used to generate card data using algorithms such as CVV, CVV2 or Dynamic CVV2. To see what keys can be used for this command, please see [Valid keys for cryptographic operations](#) section.

Many cryptographic values such as CVV, CVV2, iCVV, CAVV V7 use the same cryptographic algorithm but vary the input values. For instance [CardVerificationValue1](#) has inputs of ServiceCode, Card Number and Expiration Date. While [CardVerificationValue2](#) only has two of these inputs, this is because for CVV2/CVC2, the ServiceCode is fixed at 000. Similarly, for iCVV the ServiceCode is fixed at 999. Some algorithms may repurpose the existing fields such as CAVV V8 in which case you will need to consult your provider manual for the correct input values.

Note

Expiration date must be entered in the same format (such as MMYT vs. YYMM) for generation and validation to produce correct results.

Generate CVV2

Example

In this example, we will generate a CVV2 for a given PAN with inputs of [PAN](#) and card expiration date. This assumes that you have a card verification key [generated](#).

```
$ 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": "CADD1",  
  "ValidationData": "801"  
}
```

Generate iCVV

Example

In this example, we will generate a [iCVV](#) for a given PAN with inputs of [PAN](#), a service code of 999 and card expiration date. This assumes that you have a card verification key [generated](#).

For all available parameters see [CardVerificationValue1](#) in the API reference guide.

```
$ 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": "CADD1",
  "ValidationData": "801"
}
```

Verify card data

Verify Card Data is used to verify data that has been created using payment algorithms that rely on encryption principals such as DISCOVER_DYNAMIC_CARD_VERIFICATION_CODE.

The input values are typically provided as part of an inbound transaction to an issuer or supporting platform partner. To verify an ARQC cryptogram (used for EMV chips cards), please see [Verify ARQC](#).

For more information, see [VerifyCardValidationData](#) in the API guide.

If the value is verified, then the api will return http/200. If the value is not verified, it will return http/400.

Verify CVV2

Example

In this example, we will validate a CVV/CVV2 for a given PAN. The CVV2 is typically provided by the cardholder or user during transaction time for validation. In order to validate their input, the following values will be provided at runtime - [Key to Use for validation \(CVK\)](#), [PAN](#), card expiration date and CVV2 entered. Card expiration format must match that used in initial value generation.

For all available parameters see [CardVerificationValue2](#) in the API reference guide.

```
$ 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": "CADD1"
}
```

Verify iCVV

Example

In this example, we will verify a [iCVV](#) for a given PAN with inputs of [Key to Use for validation \(CVK\)](#), [PAN](#), a service code of 999, card expiration date and the iCVV provided by the transaction to validate.

iCVV is not a user entered value (like CVV2) but embedded on an EMV card. Consideration should be given to whether it should always validate when provided.

For all available parameters see, [CardVerificationValue1](#) in the API reference guide.

```
$ 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": "CADD1",  
  "ValidationData": "801"  
}
```

Generate, translate and verify PIN data

PIN data functions allow you to generate random pins, pin verification values (PVV) and validate inbound encrypted pins against PVV or PIN Offsets.

Pin translation allows you to translate a pin from one working key to another without exposing the pin in clear text as specified by PCI PIN Requirement 1.

Note

As PIN generation and validation are typically issuer functions and PIN translation is a typical acquirer function, we recommend that you consider least privileged access and set policies appropriately for your systems use case.

Topics

- [Translate PIN data](#)
- [Generate PIN data](#)
- [Verify PIN data](#)

Translate PIN data

Translate PIN data functions are used for translating encrypted PIN data from one set of keys to another without the encrypted data leaving the HSM. It is used for P2PE encryption where the

working keys should change but the processing system either doesn't need to, or is not permitted to, decrypt the data. The primary inputs are the encrypted data, the encryption key used to encrypt the data, the parameters used to generate the input values. The other set of inputs are the requested output parameters such as the key to be used to encrypt the output and the parameters used to create that output. The primary outputs are a newly encrypted dataset as well as the parameters used to generate it.

Note

AES key types only support ISO Format 4 [pin blocks](#).

Topics

- [PIN from PEK to DUKPT](#)
- [PIN from DUKPT to AWK](#)

PIN from PEK to DUKPT

Example

In this example, we will translate a PIN from PEK TDES encryption using ISO 0 PIN block to an AES ISO 4 PIN Block using the [DUKPT](#) algorithm. Typically this might be done in reverse, where a payment terminal encrypts a pin in ISO 4 and then it may be translated back to TDES for downstream processing.

```
$ 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-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/4pmyquwjs3yj4vwe --outgoing-translation-attributes
IsoFormat4="{PrimaryAccountNumber=171234567890123}" --outgoing-dukpt-attributes
KeySerialNumber="FFFF9876543210E00008"
```

```
{
  "PinBlock": "1F4209C670E49F83E75CC72E81B787D9",
```

```

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

```

PIN from DUKPT to AWK

Example

In this example, we will translate a PIN from an AES [DUKPT](#) encrypted PIN to a pin encrypted under a [AWK](#). It is functionally the inverse of the previous example.

```

$ aws payment-cryptography-data translate-pin-data --encrypted-pin-
block "1F4209C670E49F83E75CC72E81B787D9" --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": "AC17DC148BDA645E",
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
  "KeyCheckValue": "FE23D3"
}

```

Generate PIN data

Generate PIN data functions are used for generating PIN-related values, such as [PVV](#) and pin block offsets used for validating pin entry by users during transaction or authorization time. This API can also generate a new random pin using various algorithms.

Generate Visa PVV for a pin

Example

In this example, we will generate a new (random) pin where the outputs will be an encrypted PIN block (PinData.PinBlock) and a PVV (pinData.Offset). The key inputs are [PAN](#), the [Pin Verification Key](#), the [Pin Encryption Key](#) and the PIN block format.

This command requires that the key is of type TR31_V2_VISA_PIN_VERIFICATION_KEY.

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

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

Generate IBM3624 pin offset for a pin

IBM 3624 PIN Offset also sometimes called the IBM method. This method generates a natural/intermediate PIN using the validation data (typically the PAN) and a PIN Key (PVK). Natural pins are effectively a derived value and being deterministic are very efficient to handle for an issuer because no pin data needs to be stored at a cardholder level. The most obvious con is that this scheme doesn't account for cardholder selectable or random pins. To allow for those types of pins, an offset algorithm was added to the scheme. The offset represents the difference between the user selected (or random) pin and the natural key. The offset value is stored by the card issuer or card processor. At transaction time, the AWS Payment Cryptography service internally recalculates the natural pin and applies the offset to find the pin. It then compares this against the value provided by the transaction authorization.

Several options exist for IBM3624:

- `Ibm3624NaturalPin` will output the natural pin and an encrypted pin block
- `Ibm3624PinFromOffset` will generate an encrypted pin block given an offset
- `Ibm3624RandomPin` will generate a random pin and then the matching offset and encrypted pin block.
- `Ibm3624PinOffset` generates the pin offset given a user selected pin.

Internal to AWS Payment Cryptography, the following steps are performed:

- Pad the provided pan to 16 characters. If <16 are provided, pad on the right hand side using the provided padding character.
- Encrypts the validation data using the PIN generation key.
- Decimalize the encrypted data using the decimalization table. This maps hexadecimal digits to decimal digits for instance 'A' may map to 9 and 1 may map to 1.
- Get the first 4 digits from a hexadecimal representation of the output. This is the natural pin.
- If a user selected or random pin was generated, modulo subtract the natural pin with customer pin. The result is the pin offset.

Examples

- [Example: Generate IBM3624 pin offset for a pin](#)

Example: Generate IBM3624 pin offset for a pin

In this example, we will generate a new (random) pin where the outputs will be an encrypted PIN block (`PinData.PinBlock`) and an IBM3624 offset value (`pinData.Offset`). The inputs are [PAN](#), validation data (typically the pan), padding character, the [Pin Verification Key](#), the [Pin Encryption Key](#) and the PIN block format.

This command requires that the pin generation key is of type `TR31_V1_IBM3624_PIN_VERIFICATION_KEY` and the encryption key is of type `TR31_P0_PIN_ENCRYPTION_KEY`

Example

The following example shows generating a random pin then outputting the encrypted pin block and IBM3624 offset value using `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"
    }
}
```

Verify PIN data

Verify PIN data functions are used for verifying whether a pin is correct. This typically involves comparing the pin value previously stored against what was entered by the cardholder at a POI. These functions compare two values without exposing the underlying value of either source.

Validate encrypted PIN using PVV method

Example

In this example, we will validate a PIN for a given PAN. The PIN is typically provided by the cardholder or user during transaction time for validation and is compared against the value on file (the input from the cardholder is provided as an encrypted value from the terminal or other upstream provider). In order to validate this input, the following values will also be provided at runtime - The key used to encrypt the input pin (this is often an IWK), [PAN](#) and the value to verify against (either a PVV or PIN offset).

If AWS Payment Cryptography is able to validate the pin, an http/200 is returned. If the pin is not validated, it will return an http/400.

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

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

Validate a PIN against previously stored IBM3624 pin offset

In this example, we will validate a cardholder provided PIN against the pin offset stored on file with the card issuer/processor. The inputs are similar to [???](#) with the additional of the encrypted pin provided by the payment terminal (or other upstream provider such as card network). If the pin matches, the api will return http 200. where the outputs will be an encrypted PIN block (PinData.PinBlock) and an IBM3624 offset value (pinData.Offset).

This command requires that the pin generation key is of type TR31_V1_IBM3624_PIN_VERIFICATION_KEY and the encryption key is of type 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"
  }
}
```

Verify auth request (ARQC) cryptogram

The verify auth request cryptogram API is used for verifying [ARQC](#). The generation of the ARQC is outside of the scope of the AWS Payment Cryptography and is typically performed on an EMV Chip Card (or digital equivalent such as mobile wallet) during transaction authorization time. An ARQC is unique to each transactions and is intended to cryptographically show both the validity of the card as well as to ensure that the transaction data exactly matches the current (expected) transaction.

AWS Payment Cryptography provides a variety of options for validating ARQC and generating optional ARPC values including those defined in [EMV 4.4 Book 2](#) and other schemes used by Visa and Mastercard. For a full list of all available options, please see the [VerifyCardValidationData](#) section in the [API Guide](#).

ARQC cryptograms typically require the following inputs (although this may vary by implementation):

- [PAN](#) - Specified in the PrimaryAccountNumber field
- [PAN Sequence Number \(PSN\)](#) - specified in the PanSequenceNumber field
- Key Derivation Method such as Common Session Key (CSK) - Specified in the SessionKeyDerivationAttributes
- Master Key Derivation Mode (such as EMV Option A) - Specified in the MajorKeyDerivationMode

- Transaction data - a string of various transaction, terminal and card data such as Amount and Date - specified in the TransactionData field
- [Issuer Master Key](#) - the master key used to derive the cryptogram (AC) key used to protect individual transactions and specified in the KeyIdentifier field

Topics

- [Building transaction data](#)
- [Transaction data padding](#)
- [Examples](#)

Building transaction data

The exact content (and order) of the transaction data field varies by implementation and network scheme but the minimum recommended fields (and concatenation sequence) is defined in [EMV 4.4 Book 2 Section 8.1.1 - Data Selection](#). If the first three fields are amount (17.00), other amount (0.00) and country of purchase, that would result in the transaction data beginning as follows:

- 000000001700 - amount - 12 positions implied two digit decimal
- 000000000000 - other amount - 12 positions implied two digit decimal
- 0124 - four digit country code
- Output (partial) Transaction Data - 00000000170000000000000000124

Transaction data padding

Transaction data should be padded prior to sending to the service. Most schemes use ISO 9797 Method 2 padding, where a hex string is appended by hex 80 followed by 00 until the field is a multiple of the encryption block size; 8 bytes or 16 characters for TDES and 16 bytes or 32 characters for AES. The alternative (method 1) is not as common but uses only 00 as the padding characters.

ISO 9797 Method 1 Padding

Unpadded:

00000000170000000000000008400080008000084016051700000000093800000B03011203
(74 characters or 37 bytes)

Padded:

00000000170000000000000008400080008000084016051700000000093800000B03011203000000
(80 characters or 40 bytes)

ISO 9797 Method 2 Padding

Unpadded:

00000000170000000000000008400080008000084016051700000000093800000B1F220103000000
(80 characters or 40 bytes)

Padded:

00000000170000000000000008400080008000084016051700000000093800000B1F220103000000080
(88 characters or 44 bytes)

Examples

Visa CVN10

Example

In this example, we will validate an ARQC generated using Visa CVN10.

If AWS Payment Cryptography is able to validate the ARQC, an http/200 is returned. If the arqc is not validated, it will return a http/400 response.

```
$ aws payment-cryptography-data verify-auth-request-cryptogram --auth-request-cryptogram D791093C8A921769 \
--key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk \
--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/pw3s6n162t5ushfk",
  "KeyCheckValue": "08D7B4"
}
```

Visa CVN18 and Visa CVN22

Example

In this example, we will validate an ARQC generated using Visa CVN18 or CVN22. The cryptographic operations are the same between CVN18 and CVN22 but the data contained within transaction data varies. Compared to CVN10, a completely different cryptogram is generated even with the same inputs.

If AWS Payment Cryptography is able to validate the ARQC, an http/200 is returned. If the arqc is not validated, it will return an http/400.

```
$ aws payment-cryptography-data verify-auth-request-cryptogram \
--auth-request-cryptogram 61EDCC708B4C97B4
--key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6n162t5ushfk \
--major-key-derivation-mode EMV_OPTION_A
--transaction-data
00000000170000000000000008400080008000084016051700000000093800000B1F2201030000000000
\
000000000000000000000000000000000000000000000000000000008000000000000000
--session-key-derivation-attributes='{"EmvCommon":
{"ApplicationTransactionCounter":"000B", \
"PanSequenceNumber":"01","PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk",
  "KeyCheckValue": "08D7B4"
}
```

Generate and verify MAC

Message Authentication Codes (MAC) are typically used to authenticate the integrity of a message (whether it's been modified). Cryptographic hashes such as HMAC (Hash-Based Message Authentication Code), CBC-MAC and CMAC (Cipher-based Message Authentication Code) additionally provide additional assurance of the sender of the MAC by utilizing cryptography. HMAC is based on hash functions while CMAC is based on block ciphers.

All MAC algorithms of this service combine a cryptographic hash function and a shared secret key. They take a message and a secret key, such as the key material in a key, and return a unique tag or mac. If even one character of the message changes, or if the secret key changes, the resulting tag is entirely different. By requiring a secret key, cryptographic MACs also provides authenticity; it is impossible to generate an identical mac without the secret key. Cryptographic MACs are sometimes called symmetric signatures, because they work like digital signatures, but use a single key for both signing and verification.

AWS Payment Cryptography supports several types of MACs:

ISO9797 ALGORITHM 1

Denoted by KeyUsage of ISO9797_ALGORITHM1

ISO9797 ALGORITHM 3 (Retail MAC)

Denoted by KeyUsage of ISO9797_ALGORITHM3

ISO9797 ALGORITHM 5 (CMAC)

Denoted by KeyUsage of TR31_M6_ISO_9797_5_CMAC_KEY

HMAC

Denoted by KeyUsage of TR31_M7_HMAC_KEY including HMAC_SHA224, HMAC_SHA256, HMAC_SHA384 and HMAC_SHA512

Topics

- [Generate MAC](#)
- [Verify MAC](#)

Generate MAC

Generate MAC API is used to authenticate card-related data, such as track data from a card magnetic stripe, by using known data values to generate a MAC (Message Authentication Code) for data validation between sending and receiving parties. The data used to generate MAC includes message data, secret MAC encryption key and MAC algorithm to generate a unique MAC value for transmission. The receiving party of the MAC will use the same MAC message data, MAC encryption key, and algorithm to reproduce another MAC value for comparison and data authentication. Even

if one character of the message changes or the MAC key used for verification is not identical, the resulting MAC value is different. The API supports DUPKT MAC, HMAC and EMV MAC encryption keys for this operation.

The input value for `message-data` must be hexBinary data.

In this example, we will generate a HMAC (Hash-Based Message Authentication Code) for card data authentication using HMAC algorithm `HMAC_SHA256` and HMAC encryption key. The key must have `KeyUsage` set to `TR31_M7_HMAC_KEY` and `KeyModesOfUse` to `Generate`. The MAC key can either be created with AWS Payment Cryptography by calling [CreateKey](#) or imported by calling [ImportKey](#).

Example

```
$ aws payment-cryptography-data generate-mac \  
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/  
qnob15lghrzunce6 \  
  --message-data  
  "3b313038383439303031303733393431353d32343038323236303030373030303f33" \  
  --generation-attributes Algorithm=HMAC_SHA256
```

```
{  
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/  
qnob15lghrzunce6,  
  "KeyCheckValue": "2976E7",  
  "Mac": "ED87F26E961C6D0DDB78DA5038AA2BDDEA0DCE03E5B5E96BDDD494F4A7AA470C"  
}
```

Verify MAC

Verify MAC API is used to verify MAC (Message Authentication Code) for card-related data authentication. It must use the same encryption key used during generate MAC to re-produce MAC value for authentication. The MAC encryption key can either be created with AWS Payment Cryptography by calling [CreateKey](#) or imported by calling [ImportKey](#). The API supports DUPKT MAC, HMAC and EMV MAC encryption keys for this operation.

If the value is verified, then response parameter `MacDataVerificationSuccessful` will return `Http/200`, otherwise `Http/400` with a message indicating that Mac verification failed.

In this example, we will verify a HMAC (Hash-Based Message Authentication Code) for card data authentication using HMAC algorithm HMAC_SHA256 and HMAC encryption key. The key must have KeyUsage set to TR31_M7_HMAC_KEY and KeyModesOfUse to Verify.

Example

```
$ aws payment-cryptography-data verify-mac \  
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/  
qno151ghrzunce6 \  
  --message-data  
  "3b343038383439303031303733393431353d32343038323236303030373030303f33" \  
  --verification-attributes='Algorithm=HMAC_SHA256' \  
  --mac ED87F26E961C6D0DDB78DA5038AA2BDDEA0DCE03E5B5E96BDDD494F4A7AA470C
```

```
{  
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/  
qno151ghrzunce6,  
  "KeyCheckValue": "2976E7",  
}
```

Valid keys for cryptographic operations

Certain keys can only be used for certain operations. Additionally, some operations may limit the key modes of use for keys. Please see the following table for allowed combinations.

Note

Certain combinations, although permitted, may create unusable situations such as generating CVV codes (`generate`) but then unable to verify them (`verify`).

Topics

- [GenerateCardData](#)
- [VerifyCardData](#)
- [GeneratePinData \(for VISA/ABA schemes\)](#)
- [GeneratePinData \(for IBM3624\)](#)
- [VerifyPinData \(for VISA/ABA schemes\)](#)

- [VerifyPinData \(for IBM3624\)](#)
- [Decrypt Data](#)
- [Encrypt Data](#)
- [Translate Pin Data](#)
- [Generate/Verify MAC](#)
- [VerifyAuthRequestCryptogram](#)
- [Import/Export Key](#)
- [Unused key types](#)

GenerateCardData

API Endpoint	Cryptographic Operation or Algorithm	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
GenerateCardData	<ul style="list-style-type: none"> • AMEX_CARD_SECURITY_CODE_VERIFICATION_1 • AMEX_CARD_SECURITY_CODE_VERIFICATION_2 	TR31_C0_CARD_VERIFICATION_KEY	<ul style="list-style-type: none"> • TDES_2KEY • TDES_3KEY 	{ Generate = true }, { Generate = true, Verify = true }
GenerateCardData	<ul style="list-style-type: none"> • CARD_VERIFICATION_VALUE_1 • CARD_VERIFICATION_VALUE_2 	TR31_C0_CARD_VERIFICATION_KEY	<ul style="list-style-type: none"> • TDES_2KEY 	{ Generate = true }, { Generate = true, Verify = true }
GenerateCardData	<ul style="list-style-type: none"> • CARDHOLDER_AUTHENTICATION_V 	TR31_E6_EMV_MKEY_OTHER	<ul style="list-style-type: none"> • TDES_2KEY 	{ DeriveKey = true }

API Endpoint	Cryptographic Operation or Algorithm	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
	ERIFICATI ON_VALUE			
GenerateCardData	<ul style="list-style-type: none"> DYNAMIC_CARD_VERIFICATION_CODE 	TR31_E4_E MV_MKEY_D YNAMIC_NUMBERS	<ul style="list-style-type: none"> TDES_2KEY 	{ DeriveKey = true }
GenerateCardData	<ul style="list-style-type: none"> DYNAMIC_CARD_VERIFICATION_VALUE 	TR31_E6_E MV_MKEY_OTHER	<ul style="list-style-type: none"> TDES_2KEY 	{ DeriveKey = true }

VerifyCardData

Cryptographic Operation or Algorithm	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
<ul style="list-style-type: none"> AMEX_CARD_SECURITY_CODE_VERSION_1 AMEX_CARD_SECURITY_CODE_VERSION_2 	TR31_C0_CARD_VERIFICATION_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	{ Generate = true }, { Generate = true, Verify = true }
<ul style="list-style-type: none"> CARD_VERIFICATION_VALUE_1 CARD_VERIFICATION_VALUE_2 	TR31_C0_CARD_VERIFICATION_KEY	<ul style="list-style-type: none"> TDES_2KEY 	{ Generate = true }, { Generate = true, Verify = true }

Cryptographic Operation or Algorithm	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
• CARDHOLDER_AUTHENTICATION_VERIFICATION_VALUE	TR31_E6_E MV_MKEY_OTHER	• TDES_2KEY	{ DeriveKey = true }
• DYNAMIC_CARD_VERIFICATION_CODE	TR31_E4_E MV_MKEY_DYNAMIC_NUMBERS	• TDES_2KEY	{ DeriveKey = true }
• DYNAMIC_CARD_VERIFICATION_VALUE	TR31_E6_E MV_MKEY_OTHER	• TDES_2KEY	{ DeriveKey = true }

GeneratePinData (for VISA/ABA schemes)

VISA_PIN or VISA_PIN_VERIFICATION_VALUE

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
PIN Encryption Key	TR31_P0_P IN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY	<ul style="list-style-type: none"> • { Encrypt = true, Wrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = true }
PIN Generation Key	TR31_V2_VISA_PIN_VERIFICATION_KEY	• TDES_3KEY	• { Generate = true }

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
			<ul style="list-style-type: none"> { Generate = true, Verify = true }

GeneratePinData (for IBM3624)

IBM3624_PIN_OFFSET, IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET)

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
PIN Encryption Key	TR31_PO_P IN_ENCRYPTION_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	<p>For IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET</p> <ul style="list-style-type: none"> { Encrypt = true, Wrap = true } { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } { NoRestrictions = true } <p>For IBM3624_PIN_OFFSET</p>

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
			<ul style="list-style-type: none"> • { Encrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = true }
PIN Generation Key	TR31_V1_I BM3624_PIN_VERIFICATION_KEY	<ul style="list-style-type: none"> • TDES_3KEY 	<ul style="list-style-type: none"> • { Generate = true } • { Generate = true, Verify = true }

VerifyPinData (for VISA/ABA schemes)

VISA_PIN

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
PIN Encryption Key	TR31_P0_P IN_ENCRYPTION_KEY	<ul style="list-style-type: none"> • TDES_2KEY • TDES_3KEY 	<ul style="list-style-type: none"> • { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = true }
PIN Generation Key	TR31_V2_VISA_PIN_VERIFICATION_KEY	<ul style="list-style-type: none"> • TDES_3KEY 	<ul style="list-style-type: none"> • { Verify = true }

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
			<ul style="list-style-type: none"> { Generate = true, Verify = true }

VerifyPinData (for IBM3624)

IBM3624_PIN_OFFSET, IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET)

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
PIN Encryption Key	TR31_PO_P IN_ENCRYPTION_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	<p>For IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET</p> <ul style="list-style-type: none"> { Decrypt = true, Unwrap = true } { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } { NoRestrictions = true }
PIN Verification Key	TR31_V1_I IBM3624_PIN_VERIFICATION_KEY	<ul style="list-style-type: none"> TDES_3KEY 	<ul style="list-style-type: none"> { Verify = true } { Generate = true, Verify = true }

Decrypt Data

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
DUKPT	TR31_B0_B ASE_DERIV ATION_KEY	<ul style="list-style-type: none"> TDES_2KEY AES_128 AES_192 AES_256 	<ul style="list-style-type: none"> { DeriveKey = true } { NoRestrictions = true }
EMV	TR31_E1_E MV_MKEY_C ONFIDENTIALITY TR31_E6_E MV_MKEY_OTHER	<ul style="list-style-type: none"> TDES_2KEY 	<ul style="list-style-type: none"> { DeriveKey = true }
RSA	TR31_D1_A SYMMETRIC _KEY_FOR_ DATA_ENCRYPTION	<ul style="list-style-type: none"> RSA_2048 RSA_3072 RSA_4096 	<ul style="list-style-type: none"> { Decrypt = true, Unwrap=true} { Encrypt=true, Wrap=true, Decrypt = true, Unwrap=true }
Symmetric keys	TR31_D0_S YMMETRIC_ DATA_ENCR YPTION_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY AES_128 AES_192 AES_256 	<ul style="list-style-type: none"> { Decrypt = true, Unwrap=true} { Encrypt=true, Wrap=true, Decrypt = true, Unwrap=true } { NoRestrictions = true }

Encrypt Data

Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
DUKPT	TR31_B0_B ASE_DERIV ATION_KEY	<ul style="list-style-type: none"> TDES_2KEY AES_128 AES_192 AES_256 	<ul style="list-style-type: none"> { DeriveKey = true } { NoRestrictions = true }
EMV	TR31_E1_E MV_MKEY_C ONFIDENTIALITY TR31_E6_E MV_MKEY_OTHER	<ul style="list-style-type: none"> TDES_2KEY 	<ul style="list-style-type: none"> { DeriveKey = true }
RSA	TR31_D1_A SYMMETRIC _KEY_FOR_ DATA_ENCRYPTION	<ul style="list-style-type: none"> RSA_2048 RSA_3072 RSA_4096 	<ul style="list-style-type: none"> { Encrypt = true, Wrap=true} {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true}
Symmetric keys	TR31_D0_S YMMETRIC_ DATA_ENCR YPTION_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY AES_128 AES_192 AES_256 	<ul style="list-style-type: none"> {Encrypt = true, Wrap=true} {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true} { NoRestrictions = true }

Translate Pin Data

Direction	Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
Inbound Data Source	DUKPT	TR31_B0_B ASE_DERIV ATION_KEY	<ul style="list-style-type: none"> • TDES_2KEY • AES_128 • AES_192 • AES_256 	<ul style="list-style-type: none"> • { DeriveKey = true } • { NoRestrictions = true }
Inbound Data Source	non-DUKPT (PEK, AWK, IWK, etc)	TR31_P0_P IN_ENCRYP TION_KEY	<ul style="list-style-type: none"> • TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256 	<ul style="list-style-type: none"> • { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = true }
Outbound Data Target	DUKPT	TR31_B0_B ASE_DERIV ATION_KEY	<ul style="list-style-type: none"> • TDES_2KEY • AES_128 • AES_192 • AES_256 	<ul style="list-style-type: none"> • { DeriveKey = true } • { NoRestrictions = true }
Outbound Data Target	non-DUKPT (PEK, IWK, AWK, etc)	TR31_P0_P IN_ENCRYP TION_KEY	<ul style="list-style-type: none"> • TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256 	<ul style="list-style-type: none"> • { Encrypt = true, Wrap = true } • { Encrypt = true, Decrypt = true, Wrap = true }

Direction	Key Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
				true, Unwrap = true } • { NoRestrictions = true }

Generate/Verify MAC

MAC keys are used for creating cryptographic hashes of a message/body of data. It is not recommended to create a key with limited key modes of use as you will be unable to perform the matching operation. However, you may import/export a key with only one operation if the other system is intended to perform the other half of the operation pair.

Allowed Key Usage	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
MAC Key	TR31_M1_I SO_9797_1 _MAC_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	<ul style="list-style-type: none"> { Generate = true } { Generate = true, Verify = true } { Verify = true } { Generate = true }
MAC Key (Retail MAC)	TR31_M1_I SO_9797_3 _MAC_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	<ul style="list-style-type: none"> { Generate = true } { Generate = true, Verify = true } { Verify = true } { Generate = true }
MAC Key (CMAC)	TR31_M6_I SO_9797_5 _CMAC_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	<ul style="list-style-type: none"> { Generate = true }

Allowed Key Usage	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
		<ul style="list-style-type: none"> AES_128 AES_192 AES_256 	<ul style="list-style-type: none"> { Generate = true, Verify = true } { Verify = true } { Generate = true }
MAC Key (HMAC)	TR31_M7_HMAC_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY AES_128 AES_192 AES_256 	<ul style="list-style-type: none"> { Generate = true } { Generate = true, Verify = true } { Verify = true } { Generate = true }

VerifyAuthRequestCryptogram

Allowed Key Usage	EMV Option	Allowed Key Algorithm	Allowed combination of key modes of use
<ul style="list-style-type: none"> OPTION A OPTION B 	TR31_E0_E MV_MKEY_A PP_CRYPTOGRAMS	<ul style="list-style-type: none"> TDES_2KEY 	<ul style="list-style-type: none"> { DeriveKey = true }

Import/Export Key

Operation Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
TR-31 Wrapping Key	TR31_K1_KEY_BLOCK_PROTECTION_KEY	<ul style="list-style-type: none"> TDES_2KEY TDES_3KEY 	<ul style="list-style-type: none"> { Encrypt = true, Wrap = true } (export only)

Operation Type	Allowed Key Usage	Allowed Key Algorithm	Allowed combination of key modes of use
	TR31_K0_KEY_ENCRYPTION_KEY	<ul style="list-style-type: none"> AES_128 	<ul style="list-style-type: none"> { Decrypt = true, Unwrap = true } (import only) { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }
Import of trusted CA	TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE	<ul style="list-style-type: none"> RSA_2048 RSA_3072 RSA_4096 	<ul style="list-style-type: none"> { Verify = true }
Import of public key certificate for asymmetric encryption	TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION	<ul style="list-style-type: none"> RSA_2048 RSA_3072 RSA_4096 	<ul style="list-style-type: none"> { Encrypt=true, Wrap=true }

Unused key types

The following key types are not currently used by AWS Payment Cryptography

- TR31_P1_PIN_GENERATION_KEY
- TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT

Common use cases

AWS Payment Cryptography supports many typical payment cryptographic operations. The following topics act as a guide on how to use these operations for typical common use cases. For a list of all commands, please review the AWS Payment Cryptography API.

Topics

- [Issuers and issuer processors](#)
- [Acquiring and payment facilitators](#)

Issuers and issuer processors

Issuer use cases typically consist of a few parts. This section is organized by function (such as working with pins). In a production system, the keys are typically scoped to a given card bin and are created during bin setup rather than inline as shown here.

Topics

- [General Functions](#)
- [Network specific functions](#)

General Functions

Topics

- [Generate a random pin and the associated PVV and then verify the value](#)
- [Generate or verify a CVV for a given card](#)
- [Generate or verify a CVV2 for a specific card](#)
- [Generate or verify a iCVV for a specific card](#)
- [Verify an EMV ARQC and generate an ARPC](#)
- [Generate and Verify an EMV MAC](#)

Generate a random pin and the associated PVV and then verify the value

Topics

- [Create the key\(s\)](#)
- [Generate a random pin, generate PVV and return the encrypted PIN and PVV](#)
- [Validate encrypted PIN using PVV method](#)

Create the key(s)

In order to generate a random pin and the [PVV](#), you'll need two keys, a [Pin Verification Key\(PVK\)](#) for generating the PVV and a [Pin Encryption Key](#) for encrypting the pin. The pin itself is randomly generated securely inside the service and is not related to either key cryptographically.

The PGK must be a key of algorithm TDES_2KEY based on the PVV algorithm itself. A PEK can be TDES_2KEY, TDES_3KEY or AES_128. In this case, since the PEK is intended for internal use within your system, AES_128 would be a good choice. If a PEK is used for interchange with other systems (e.g. card networks, acquirers, ATMs) or are being moved as part of a migration, TDES_2KEY may be the more appropriate choice for compatibility reasons.

Create the PEK

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

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

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

```

        "Wrap": false,
        "Unwrap": false,
        "Generate": true,
        "Sign": false,
        "Verify": true,
        "DeriveKey": false,
        "NoRestrictions": false
    }
},
"KeyCheckValue": "7CC9E2",
"KeyCheckValueAlgorithm": "CMAC",
"Enabled": true,
"Exportable": true,
"KeyState": "CREATE_COMPLETE",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
"UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
}
}

```

Take note of the `KeyArn` that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt`. You need that in the next step.

Create the PVK

```

$ aws payment-cryptography create-key --exportable --key-attributes
  KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_V2_VISA_PIN_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyMo
  --tags='[{"Key":"CARD_BIN","Value":"12345678"}]'

```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```

{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza",
        "KeyAttributes": {
            "KeyUsage": "TR31_V2_VISA_PIN_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_2KEY",
            "KeyModesOfUse": {
                "Encrypt": false,

```

```

        "Decrypt": false,
        "Wrap": false,
        "Unwrap": false,
        "Generate": true,
        "Sign": false,
        "Verify": true,
        "DeriveKey": false,
        "NoRestrictions": false
    }
},
"KeyCheckValue": "51A200",
"KeyCheckValueAlgorithm": "ANSI_X9_24",
"Enabled": true,
"Exportable": true,
"KeyState": "CREATE_COMPLETE",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
"UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
}
}

```

Take note of the KeyArn that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/ov6icy4ryas4zcza`. You need that in the next step.

Generate a random pin, generate PVV and return the encrypted PIN and PVV

Example

In this example, we will generate a new (random) 4 digit pin where the outputs will be an encrypted PIN block (PinData.PinBlock) and a PVV (pinData.VerificationValue). The key inputs are [PAN](#), the [Pin Verification Key](#) (also known as the pin generation key), the [Pin Encryption Key](#) and the [PIN Block](#) format.

This command requires that the key is of type TR31_V2_VISA_PIN_VERIFICATION_KEY.

```

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

```

```
{
```

```

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

```

Validate encrypted PIN using PVV method

Example

In this example, we will validate a PIN for a given PAN. The PIN is typically provided by the cardholder or user during transaction time for validation and is compared against the value on file (the input from the cardholder is provided as an encrypted value from the terminal or other upstream provider). In order to validate this input, the following values will also be provided at runtime - The encrypted pin, the key used to encrypt the input pin (often referred to as an [IWK](#)), [PAN](#) and the value to verify against (either a PVV or PIN offset).

If AWS Payment Cryptography is able to validate the pin, an http/200 is returned. If the pin is not validated, it will return an http/400.

```

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

```

```

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

```

```
}
```

Generate or verify a CVV for a given card

[CVV](#) or CVV1 is a value that is traditionally embedded in a cards magnetic stripe. It is not the same as CVV2 (visible to the cardholder and for use for online purchases).

The first step is to create a key. For this tutorial, you create a [CVK](#) double-length 3DES (2KEY TDES) key.

Note

CVV, CVV2 and iCVV all use similar if not identical algorithms but vary the input data. All use the same key type TR31_C0_CARD_VERIFICATION_KEY but it is recommended to use separate keys for each purpose. These can be distinguished using aliases and/or tags as in the example below.

Create the key

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=ENCRYPT,DECRYPT,WRAP,UNWRAP,GENERATE
--tags='[{"Key":"KEY_PURPOSE","Value":"CVV"}, {"Key":"CARD_BIN","Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
r52o3wbqxyf6qlqr",
    "KeyAttributes": {
      "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDES_2KEY",
      "KeyModesOfUse": {
        "Encrypt": false,
        "Decrypt": false,
        "Wrap": false,
        "Unwrap": false,
        "Generate": true,

```



```

        "Sign": false,
        "Verify": true,
        "DeriveKey": false,
        "NoRestrictions": false
    }
},
"KeyCheckValue": "DE89F9",
"KeyCheckValueAlgorithm": "ANSI_X9_24",
"Enabled": true,
"Exportable": true,
"KeyState": "CREATE_COMPLETE",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
"UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
}
}

```

Take note of the KeyArn that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/r52o3wbqxyf6qlqr`. You need that in the next step.

Generate a CVV

Example

In this example, we will generate a [CVV](#) for a given PAN with inputs of [PAN](#), a service code (as defined by ISO/IEC 7813) of 121 and card expiration date.

For all available parameters see [CardVerificationValue1](#) in the API reference guide.

```

$ 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"
}

```

Validate CVV

Example

In this example, we will verify a [CVV](#) for a given PAN with inputs of an CVK, [PAN](#), a service code of 121, card expiration date and the CVV provided during the transaction to validate.

For all available parameters see, [CardVerificationValue1](#) in the API reference guide.

Note

CVV is not a user entered value (like CVV2) but is typically embedded on a magstripe. Consideration should be given to whether it should always validate when provided.

```
$ 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"
}
```

Generate or verify a CVV2 for a specific card

[CVV2](#) is a value that is traditionally provided on the back of a card and is used for online purchases. For virtual cards, it might also be displayed on an app or a screen. Cryptographically, it is the same as CVV1 but with a different service code value.

Create the key

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModes0
--tags=' [{"Key":"KEY_PURPOSE", "Value":"CVV2"}, {"Key":"CARD_BIN", "Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/7f7g4spf3xcklhzu",
    "KeyAttributes": {
      "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDES_2KEY",
      "KeyModesOfUse": {
        "Encrypt": false,
        "Decrypt": false,
        "Wrap": false,
        "Unwrap": false,
        "Generate": true,
        "Sign": false,
        "Verify": true,
        "DeriveKey": false,
        "NoRestrictions": false
      }
    },
    "KeyCheckValue": "AEA5CD",
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "Enabled": true,
    "Exportable": true,
    "KeyState": "CREATE_COMPLETE",
    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
    "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
    "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
  }
}
```

Take note of the KeyArn that represents the key, for example *arn:aws:payment-cryptography:us-east-2:111122223333:key/7f7g4spf3xcklhzu*. You need that in the next step.

Generate a CVV2

Example

In this example, we will generate a [CVV2](#) for a given PAN with inputs of [PAN](#) and card expiration date.

For all available parameters see [CardVerificationValue2](#) in the API reference guide.

```
$ 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"
}
```

Validate a CVV2

Example

In this example, we will verify a [CVV2](#) for a given PAN with inputs of an CVK, [PAN](#) and card expiration date and the CVV provided during the transaction to validate.

For all available parameters see, [CardVerificationValue2](#) in the API reference guide.

Note

CVV2 and the other inputs are user entered values. As such, it is not necessarily a sign of an issue that this periodically fails to validate.

```
$ 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"
  }

```

Generate or verify a iCVV for a specific card

[iCVV](#) uses the same algorithm as CVV/CVV2 but iCVV is embedded inside a chip card. Its service code is 999.

Create the key

```

$ aws payment-cryptography create-key --exportable --key-attributes
  KeyAlgorithm=TDDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=GENERATE_VERIFY,
  --tags='[{"Key":"KEY_PURPOSE","Value":"ICVV"}, {"Key":"CARD_BIN","Value":"12345678"}]'

```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```

{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/c7dsi763r6s7lfp3",
    "KeyAttributes": {
      "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDDES_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"
    }
}

```

Take note of the `KeyArn` that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/c7dsi763r6s7lfp3`. You need that in the next step.

Generate a iCVV

Example

In this example, we will generate a [iCVV](#) for a given PAN with inputs of [PAN](#), a service code (as defined by ISO/IEC 7813) of 999 and card expiration date.

For all available parameters see [CardVerificationValue1](#) in the API reference guide.

```

$ 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"
}

```

Validate iCVV

Example

For validation, the inputs are CVK, [PAN](#), a service code of 999, card expiration date and the iCVV provided during the transaction to validate.

For all available parameters see, [CardVerificationValue1](#) in the API reference guide.

Note

iCVV is not a user entered value (like CVV2) but is typically embedded on an EMV/chip card. Consideration should be given to whether it should always validate when provided.

```
$ 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"
}
```

Verify an EMV ARQC and generate an ARPC

[ARQC](#) (Authorization Request Cryptogram) is a cryptogram generated by an EMV (chip) card and used to validate the transaction details as well as the use of an authorized card. It incorporates data from the card, terminal and the transaction itself.

At validation time on the backend, the same inputs are provided to AWS Payment Cryptography, the cryptogram is internally re-created and this is compared against the value provided with the transaction. In this sense, it is similar to a MAC. [EMV 4.4 Book 2](#) defines three aspects of this function - key derivation methods (known as common session key - CSK) to generate one-time transaction keys, a minimum payload and methods for generating a response (ARPC).

Individual card schemes may specify additional transactional fields to incorporate or the order those fields appear. Other (generally deprecated) scheme specific derivation schemes exist as well and are covered elsewhere in this documentation.

For more information, see [VerifyCardValidationData](#) in the API guide.

Create the key

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDDES_2KEY,KeyUsage=TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS,KeyClass=SYMMETRIC_KEY,KeyMod
--tags='[{"Key":"KEY_PURPOSE","Value":"CVN18"}, {"Key":"CARD_BIN","Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyAttributes": {
      "KeyUsage": "TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDDES_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"
  }
}
```

Take note of the `KeyArn` that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk`. You need that in the next step.

Generate an ARQC

The ARQC is generated exclusively by an EMV card. As such, AWS Payment Cryptography has no facility for generating such a payload. For test purposes, a number of libraries are available online that can generate an appropriate payload as well as known values that are generally provided by the various schemes.

Validate an ARQC

Example

If AWS Payment Cryptography is able to validate the ARQC, an http/200 is returned. An ARPC (response) can optionally be provided and included in the response after the ARQC is validated.

```
$ aws payment-cryptography-data verify-auth-request-cryptogram
--auth-request-cryptogram 61EDCC708B4C97B4 --key-identifier
arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk
--major-key-derivation-mode EMV_OPTION_A --transaction-data
0000000017000000000000000000008400080008000084016051700000000093800000B1F2201030000000000000000000000
--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/
pw3s6n162t5ushfk",
  "KeyCheckValue": "08D7B4",
  "AuthResponseValue": "2263AC85"
}
```

Generate and Verify an EMV MAC

EMV MAC is MAC using an input of an EMV derived key and then performing a ISO9797-3 (Retail) MAC over the resulting data. EMV MAC is typically used to send commands to an EMV card such as unblock scripts.

Note

AWS Payment Cryptography does not validate the contents of the script. Please consult your scheme or card manual for details on specific commands to include.

For more information, see [MacAlgorithmEmv](#) in the API guide.

Topics

- [Create the key](#)
- [Generate an EMV MAC](#)

Create the key

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E2_EMV_MKEY_INTEGRITY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=
--tags=' [{"Key":"KEY_PURPOSE","Value":"CVN18"}, {"Key":"CARD_BIN","Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6n162t5ushfk",
    "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"  
    }  
}
```

Take note of the `KeyArn` that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk`. You need that in the next step.

Generate an EMV MAC

The typical flow is that a backend process will generate an EMV script (such as card unblock), sign it using this command (which derives a one-time key specific to one particular card) and then return the MAC. Then the command + MAC are sent to the card to be applied. Sending the command to the card is outside the scope of AWS Payment Cryptography.

Note

This command is meant for commands when no encrypted data (such as PIN) is sent. EMV Encrypt can be combined with this command to append encrypted data to the issuer script prior to calling this command

Message Data

Message data includes the APDU header and command. While this can vary by implementation, this example is the APDU header for unblock (84 24 00 00 08), following by ATC (0007) and then ARQC of the previous transaction (999E57FD0F47CACE). The service does not validate the contents of this field.

Session Key Derivation Mode

This field defines how the session key is generated. `EMV_COMMON_SESSION_KEY` is generally used for the new implementations, while `EMV2000 | AMEX | MASTERCARD_SESSION_KEY | VISA` may be used as well.

MajorKeyDerivationMode

EMV Defines Mode A, B or C. Mode A is the most common and AWS Payment Cryptography currently supports mode A or mode B.

PAN

The account number, typically available in chip field 5A or ISO8583 field 2 but may also be retrieved from the card system.

PSN

The card sequence number. If not used, enter 00.

SessionKeyDerivationValue

This is the per session derivation data. It can either be the last ARQC(ApplicationCryptogram) from field 9F26 or the last ATC from 9F36 depending on the derivation scheme.

Padding

Padding is automatically applied and uses ISO/IEC 9797-1 padding method 2.

Example

```
$ aws payment-cryptography-data generate-mac --message-data
84240000080007999E57FD0F47CACE --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk --message-
data 8424000008999E57FD0F47CACE0007 --generation-attributes
EmvMac="{MajorKeyDerivationMode=EMV_OPTION_A,PanSequenceNumber='00',PrimaryAccountNumber='2235
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk",
  "KeyCheckValue": "08D7B4",
  "Mac": "5652EEDF83EA0D84"
}
```

Network specific functions

Topics

- [Visa specific functions](#)
- [Mastercard specific functions](#)
- [American Express specific functions](#)

Visa specific functions

Topics

- [ARQC - CVN18/CVN22](#)
- [ARQC - CVN10](#)
- [CAVV V7](#)

ARQC - CVN18/CVN22

CVN18 and CVN22 utilize the [CSK method](#) of key derivation. The exact transaction data varies between these two methods - please see the scheme documentation for details on constructing the transaction data field.

ARQC - CVN10

CVN10 is an older Visa method for EMV transactions that uses per card key derivation rather than session (per transaction) derivation and also uses a different payload. For information on the payload contents, please contact the scheme for details.

Create key

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDDES_2KEY,KeyUsage=TR31_E0_EMV_MKEY_APP_CRYPTGRAMS,KeyClass=SYMMETRIC_KEY,KeyMod
--tags='[{"Key":"KEY_PURPOSE","Value":"CVN10"}, {"Key":"CARD_BIN","Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/pw3s6nl62t5ushfk",
        "KeyAttributes": {
            "KeyUsage": "TR31_E0_EMV_MKEY_APP_CRYPTGRAMS",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDDES_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"
}
}

```

Take note of the KeyArn that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk`. You need that in the next step.

Validate the ARQC

Example

In this example, we will validate an ARQC generated using Visa CVN10.

If AWS Payment Cryptography is able to validate the ARQC, an http/200 is returned. If the arqc is not validated, it will return a http/400 response.

```

$ 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
000000001700000000000000000000840008000084016051700000000093800000B03011203000000 \
    --session-key-derivation-attributes='{"Visa":{"PanSequenceNumber":"01" \
, "PrimaryAccountNumber":"9137631040001422"}}'

```

```
{
```

```

    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6n162t5ushfk",
    "KeyCheckValue": "08D7B4"
  }

```

CAVV V7

For Visa Secure (3DS) transactions, a CAVV (Cardholder Authentication Verification Value) is generated by the issuer Access Control Server (ACS). The CAVV is evidence that cardholder authentication occurred, is unique for each authentication transaction and is provided by the acquirer in the authorization message. CAVV v7 binds additional data about the transaction to the approval including elements such as merchant name, purchase amount and purchase date. In this way, it is effectively a cryptographic hash of the transaction payload.

Cryptographically, CAVV V7 utilizes the CVV algorithm but the inputs have all been changed/repurposed. Please consult appropriate third party/Visa documentation on how to produce the inputs to generate a CAVV V7 payload.

Create the key

```

$ aws payment-cryptography create-key --exportable --key-attributes
  KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=ENCRYPT,DECRYPT,UNWRAP,GENERATE
  --tags=' [{"Key":"KEY_PURPOSE","Value":"CAVV-V7"},
{"Key":"CARD_BIN","Value":"12345678"}]'

```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```

{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/dnaeyrjgdjttw6dk",
    "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"
}
}

```

Take note of the KeyArn that represents the key, for example *arn:aws:payment-cryptography:us-east-2:111122223333:key/dnaeyrjgdjttw6dk*. You need that in the next step.

Generate a CAVV V7

Example

In this example, we will generate a CAVV V7 for a given transactions with inputs as specified in the specifications. Note that for this algorithm, fields may be re-used/re-purposed, so it should not be assumed that the field labels match the inputs.

For all available parameters see [CardVerificationValue1](#) in the API reference guide.

```

$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjttw6dk --primary-account-number=171234567890123 --generation-attributes
CardVerificationValue1='{CardExpiryDate=9431,ServiceCode=431}'

```

```

{
  "KeyArn": "",
  "KeyCheckValue": "F3FB13",
  "ValidationData": "491"
}

```


Validate CAVV V7

Example

For validation, the inputs are CVK, the computed input values and the CAVV provided during the transaction to validate.

For all available parameters see, [CardVerificationValue1](#) in the API reference guide.

Note

CAVV is not a user entered value (like CVV2) but is calculated by the issuer ACS. Consideration should be given to whether it should always validate when provided.

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
arn:aws:payment-cryptography:us-east-2:111122223333:key/dnaeyrjgdjttw6dk
--primary-account-number=171234567890123 --verification-attributes
CardVerificationValue1='{CardExpiryDate=9431,ServiceCode=431} --validation-data 491
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/dnaeyrjgdjttw6dk",
    "KeyCheckValue": "F3FB13",
    "ValidationData": "491"
}
```

Mastercard specific functions

Topics

- [DCVC3](#)
- [ARQC - CVN14/CVN15](#)
- [ARQC - CVN12/CVN13](#)

DCVC3

DCVC3 predates EMV CSK and Mastercard CVN12 schemes and represents another approach for utilizing dynamic keys. It is sometimes repurposed for other use cases as well. In this scheme, the inputs are PAN, PSN, Track1/Track2 data, an unpredictable number and transaction counter (ATC).

Create key

```
$ 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"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "",
    "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": "",
    "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"
  }
}
```

```
}

```

Take note of the KeyArn that represents the key, for example . You need that in the next step.

Generate a DCVC3

Example

Although DCVC3 may be generated by a chip card, it can also be manually generated such as in this example

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier
arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk
--primary-account-number=5413123456784808 --generation-attributes
DynamicCardVerificationCode='{ApplicationTransactionCounter=0000,TrackData=5241060000000069D13
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6n162t5ushfk",
    "KeyCheckValue": "08D7B4",
    "ValidationData": "865"
}
```

Validate the DCVC3

Example

In this example, we will validate an DCVC3. Note that ATC should be provided as a hex number for instance a counter of 11 should be represented as 000B. The service expects a 3 digit DCVC3, so if you have stored a 4(or 5) digit value, simply truncate the left characters until you have 3 digits (for instance 15321 should result in validation-data value of 321).

If AWS Payment Cryptography is able to validate, an http/200 is returned. If the value is not validated, it will return a http/400 response.

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6n162t5ushfk
--primary-account-number=5413123456784808 --verification-attributes
DynamicCardVerificationCode='{ApplicationTransactionCounter=000B,TrackData=5241060000000069D13
--validation-data 398
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6n162t5ushfk",
    "KeyCheckValue": "08D7B4"
}
```

ARQC - CVN14/CVN15

CVN14 and CVN15 utilize the [EMV CSK method](#) of key derivation. The exact transaction data varies between these two methods - please see the scheme documentation for details on constructing the transaction data field.

ARQC - CVN12/CVN13

CVN12 and CVN13 are older Mastercard-specific method for EMV transactions that incorporates an unpredictable number into the per-transaction derivation and also uses a different payload. For information on the payload contents, please contact the scheme.

Create key

```
$ 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"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/pw3s6n162t5ushfk",
        "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"
}
}

```

Take note of the KeyArn that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk`. You need that in the next step.

Validate the ARQC

Example

In this example, we will validate an ARQC generated using Mastercard CVN12.

If AWS Payment Cryptography is able to validate the ARQC, an http/200 is returned. If the arqc is not validated, it will return a http/400 response.

```

$ 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 0000000015000000000000000084000000000008402312120197695905
\
    --session-key-derivation-attributes='{"Mastercard":{"PanSequenceNumber":"01"
\
, "PrimaryAccountNumber":"9137631040001422", "ApplicationTransactionCounter":"000B", "Unpredictab

```

```

{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",

```

```
    "KeyCheckValue": "08D7B4"
  }
```

American Express specific functions

Topics

- [CSC1](#)
- [CSC2](#)

CSC1

CSC Version 1 is also known as the Classic CSC Algorithm. The service can provide it as a 3,4 or 5 digit number.

For all available parameters see [AmexCardSecurityCodeVersion1](#) in the API reference guide.

Create key

```
$ aws payment-cryptography create-key --exportable --key-attributes
  KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=ENCRYPT,DECRYPT,WRAP,UNWRAP,GENERATE,SIGN,VERIFY
  --tags=' [{"Key":"KEY_PURPOSE","Value":"CSC1"}, {"Key":"CARD_BIN","Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/esh6hn7pxdtttzgq",
    "KeyAttributes": {
      "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDES_2KEY",
      "KeyModesOfUse": {
        "Encrypt": false,
        "Decrypt": false,
        "Wrap": false,
        "Unwrap": false,
        "Generate": true,
        "Sign": false,
        "Verify": true,
      }
    }
  }
}
```

```

        "DeriveKey": false,
        "NoRestrictions": false
    }
},
"KeyCheckValue": "8B5077",
"KeyCheckValueAlgorithm": "ANSI_X9_24",
"Enabled": true,
"Exportable": true,
"KeyState": "CREATE_COMPLETE",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
"UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
}
}

```

Take note of the KeyArn that represents the key, for example *arn:aws:payment-cryptography:us-east-2:111122223333:key/esh6hn7pxdtttzgq*. You need that in the next step.

Generate a CSC1

Example

```

$ aws payment-cryptography-data generate-card-validation-data --key-
  identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
  esh6hn7pxdtttzgq --primary-account-number=344131234567848 --generation-attributes
  AmexCardSecurityCodeVersion1='{CardExpiryDate=1224}' --validation-data-length 4

```

```

{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
  esh6hn7pxdtttzgq",
    "KeyCheckValue": "8B5077",
    "ValidationData": "3938"
}

```

Validate the CSC1

Example

In this example, we will validate a CSC1.

If AWS Payment Cryptography is able to validate, an http/200 is returned. If the value is not validated, it will return a http/400 response.

```
$ 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 Version 2 is also known as the Enhanced CSC Algorithm. The service can provide it as a 3,4 or 5 digit number.

For all available parameters see [AmexCardSecurityCodeVersion2](#) in the API reference guide.

Create key

```
$ aws payment-cryptography create-key --exportable --key-attributes
KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse=ENCRYPT,DECRYPT,WRAP,UNWRAP,GENERATE,SIGN
--tags='[{"Key":"KEY_PURPOSE","Value":"CSC1"}, {"Key":"CARD_BIN","Value":"12345678"}]'
```

The response echoes back the request parameters, including an ARN for subsequent calls as well as a Key Check Value (KCV).

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
esh6hn7pxdtttzgq",
    "KeyAttributes": {
      "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "TDES_2KEY",
      "KeyModesOfUse": {
        "Encrypt": false,
        "Decrypt": false,
        "Wrap": false,
        "Unwrap": false,
        "Generate": true,
        "Sign": false,

```



```

        "Verify": true,
        "DeriveKey": false,
        "NoRestrictions": false
    }
},
"KeyCheckValue": "8B5077",
"KeyCheckValueAlgorithm": "ANSI_X9_24",
"Enabled": true,
"Exportable": true,
"KeyState": "CREATE_COMPLETE",
"KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
"CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
"UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
}
}

```

Take note of the KeyArn that represents the key, for example `arn:aws:payment-cryptography:us-east-2:111122223333:key/esh6hn7pxdtttzgg`. You need that in the next step.

Generate a CSC2

In this example, we will generate a CSC2 with a length of 5. CSC can be generated with a length of 3,4 or 5. For American Express, PANs should be 15 digits and start with 34 or 37.

Example

```

$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
esh6hn7pxdtttzgg --primary-account-number=344131234567848 --generation-attributes
  AmexCardSecurityCodeVersion2='{CardExpiryDate=1224,ServiceCode=999}' --validation-
data-length 4

```

```

{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/erlm445qvunmvoda",
  "KeyCheckValue": "BF1077",
  "ValidationData": "3982"
}

```

Validate the CSC2

Example

In this example, we will validate a CSC2.

If AWS Payment Cryptography is able to validate, an http/200 is returned. If the value is not validated, it will return a http/400 response.

```
$ 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=1224,ServiceCode=999}' --validation-data
3982
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/erlm445qvunmvoda",
  "KeyCheckValue": "BF1077"
}
```

Acquiring and payment facilitators

Acquirers, PSPs and Payment Facilitators typically have a different set of cryptographic requirements than issuers. Common use cases include:

Data Decryption

Data (especially pan data) may be encrypted by a payment terminal and need to be decrypted by the backend. [Decrypt Data](#) and Encrypt Data support a variety of methods including TDES, AES and DUKPT derivation techniques. The AWS Payment Cryptography service itself is also PCI P2PE compliant and is registered as a PCI P2PE decryption component.

TranslatePin

To maintain PCI PIN compliance, acquiring systems shall not have cardholder pins in the clear after they have been entered on a secure device. Therefore, to pass the pin onward from terminal to a downstream system (such as a payment network or issuer), there is a need to re-encrypt it using a different key than the one that the payment terminal used. [Translate Pin](#) accomplishes that by converting an encrypted pin from one key to another securely with the servicebbb. Using this command, pins can be converted between various schemes such as TDES, AES and DUKPT derivation and pin block formats such as ISO-0, ISO-3 and ISO-4.

VerifyMac

Data from a payment terminal may be MAC'd to ensure that the data hasn't been modified in transit. [Verify Mac](#) and GenerateMac supports a variety of techniques using symmetric keys

including TDES, AES and DUKPT derivation techniques for use with ISO-9797-1 algorithm 1, ISO-9797-1 algorithm 3 (Retail MAC) and CMAC techniques.

Additional Topics

- [Using Dynamic Keys](#)

Using Dynamic Keys

Dynamic Keys allows one-time or limited use keys to be used for cryptographic operations like [EncryptData](#). This flow can be utilized when the key material frequently rotates (such as on every card transaction) and there is a desire to avoid importing the key material into the service. Short-lived keys may be utilized as part of [softPOS/Mpoc](#) or other solutions.

Note

This can be used in lieu of the typical flow using AWS Payment Cryptography, where cryptographic keys are either created or imported into the service and keys are specified using a key alias or key arn.

The following operations support Dynamic Keys:

- EncryptData
- DecryptData
- ReEncryptData
- TranslatePin

Decrypting Data

The following example shows using Dynamic Keys along with the decrypt command. The key identifier in this case is the wrapping key (KEK) that secures the decryption key (that is provided in the wrapped-key parameter in TR-31 format). The wrapped key shall be key purpose of D0 to be used with decrypt command along with a mode of use of B or D.

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"
}
```

Translating a pin

The following example shows using Dynamic Keys along with the translate pin command to translate from a dynamic key to a semi-static acquirer working key (AWK). The incoming key identifier in this case is the wrapping key (KEK) that is protecting the dynamic pin encryption key (PEK) that is provided in the TR-31 format. The wrapped key shall be key purpose of P0 along with a mode of use of B or D. The outgoing key identifier is a key of type TR31_P0_PIN_ENCRYPTION_KEY and a mode of use of Encrypt=true,Wrap=true

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"  
}
```

Security in AWS Payment Cryptography

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security in the cloud:

- **Security of the cloud**—AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS Payment Cryptography, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud**—Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This topic helps you understand how to apply the shared responsibility model when using AWS Payment Cryptography. It shows you how to configure AWS Payment Cryptography to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Payment Cryptography resources.

Topics

- [Data protection in AWS Payment Cryptography](#)
- [Resilience in AWS Payment Cryptography](#)
- [Infrastructure security in AWS Payment Cryptography](#)
- [Connecting to AWS Payment Cryptography through a VPC endpoint](#)
- [Security best practices for AWS Payment Cryptography](#)

Data protection in AWS Payment Cryptography

The AWS [shared responsibility model](#) applies to data protection in AWS Payment Cryptography. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on

this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS Payment Cryptography or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

AWS Payment Cryptography stores and protects your payment encryption keys to make them highly available while providing you with strong and flexible access control.

Topics

- [Protecting key material](#)
- [Data encryption](#)

- [Encryption at rest](#)
- [Encryption in transit](#)
- [Internetwork traffic privacy](#)

Protecting key material

By default, AWS Payment Cryptography protects the cryptographic key material for payment keys managed by the service. In addition, AWS Payment Cryptography offers options for importing key material that is created outside of the service. For technical details about payment keys and key material, see [AWS Payment Cryptography Cryptographic Details](#).

Data encryption

The data in AWS Payment Cryptography consists of AWS Payment Cryptography keys, the encryption key material they represent, and their usage attributes. Key material exists in plaintext only within AWS Payment Cryptography hardware security modules (HSMs) and only when in use. Otherwise, the key material and attributes are encrypted and stored in durable persistent storage.

The key material that AWS Payment Cryptography generates or loads for payment keys never leaves the boundary of AWS Payment Cryptography HSMs unencrypted. It can be exported encrypted by AWS Payment Cryptography API operations.

Encryption at rest

AWS Payment Cryptography generates key material for payment keys in PCI PTS HSM-listed HSMs. When not in use, key material is encrypted by an HSM key and written to durable, persistent storage. The key material for Payment Cryptography keys and the encryption keys that protect the key material never leave the HSMs in plaintext form.

Encryption and management of key material for Payment Cryptography keys is handled entirely by the service.

For more details, see [AWS Key Management Service Cryptographic Details](#).

Encryption in transit

Key material that AWS Payment Cryptography generates or loads for payment keys is never exported or transmitted in AWS Payment Cryptography API operations in cleartext. AWS Payment Cryptography uses key identifiers to represent the keys in API operations.

However, some AWS Payment Cryptography API operations export keys encrypted by a previously shared or asymmetric key exchange key. Also, customers can use API operations to import encrypted key material for payment keys.

All AWS Payment Cryptography API calls must be signed and be transmitted using Transport Layer Security (TLS). AWS Payment Cryptography requires TLS versions and cipher suites defined by PCI as "strong cryptography". All service endpoints support TLS 1.0—1.3 and hybrid post-quantum TLS.

For more details, see [AWS Key Management Service Cryptographic Details](#).

Internetwork traffic privacy

AWS Payment Cryptography supports an AWS Management Console and a set of API operations that enable you to create and manage payment keys and use them in cryptographic operations.

AWS Payment Cryptography supports two network connectivity options from your private network to AWS.

- An IPsec VPN connection over the internet.
- AWS Direct Connect, which links your internal network to an AWS Direct Connect location over a standard Ethernet fiber-optic cable.

All Payment Cryptography API calls must be signed and be transmitted using Transport Layer Security (TLS). The calls also require a modern cipher suite that supports perfect forward secrecy. Traffic to the hardware security modules (HSMs) that store key material for payment keys is permitted only from known AWS Payment Cryptography API hosts over the AWS internal network.

To connect directly to AWS Payment Cryptography from your virtual private cloud (VPC) without sending traffic over the public internet, use VPC endpoints, powered by AWS PrivateLink. For more information, see [Connecting to AWS Payment Cryptography through a VPC endpoint](#).

AWS Payment Cryptography also supports a hybrid post-quantum key exchange option for the Transport Layer Security (TLS) network encryption protocol. You can use this option with TLS when you connect to AWS Payment Cryptography API endpoints.

Resilience in AWS Payment Cryptography

AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through

low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Regional isolation

AWS Payment Cryptography is a Regional service that is available in multiple regions.

The Regionally isolated design of AWS Payment Cryptography ensures that an availability issue in one AWS Region cannot affect AWS Payment Cryptography operation in any other Region. AWS Payment Cryptography is designed to ensure zero planned downtime, with all software updates and scaling operations performed seamlessly and imperceptibly.

The AWS Payment Cryptography Service Level Agreement (SLA) includes a service commitment of 99.99% for all Payment Cryptography APIs. To fulfill this commitment, AWS Payment Cryptography ensures that all data and authorization information required to execute an API request is available on all regional hosts that receive the request.

The AWS Payment Cryptography infrastructure is replicated in at least three Availability Zones (AZs) in each Region. To ensure that multiple host failures do not affect AWS Payment Cryptography performance, AWS Payment Cryptography is designed to service customer traffic from any of the AZs in a Region.

Changes that you make to the properties or permissions of a payment key are replicated to all hosts in the Region to ensure that subsequent request can be processed correctly by any host in the Region. Requests for cryptographic operations using your payment key are forwarded to a fleet of AWS Payment Cryptography hardware security modules (HSMs), any of which can perform the operation with the payment key.

Multi-tenant design

The multi-tenant design of AWS Payment Cryptography enables it to fulfill the availability SLA, and to sustain high request rates, while protecting the confidentiality of your keys and data.

Multiple integrity-enforcing mechanisms are deployed to ensure that the payment key that you specified for the cryptographic operation is always the one that is used.

The plaintext key material for your Payment Cryptography keys is protected extensively. The key material is encrypted in the HSM as soon as it is created, and the encrypted key material is immediately moved to secure storage. The encrypted key is retrieved and decrypted within the HSM just in time for use. The plaintext key remains in HSM memory only for the time needed to complete the cryptographic operation. Plaintext key material never leaves the HSMs; it is never written to persistent storage.

For more information about the mechanisms that AWS Payment Cryptography uses to secure your keys, see [AWS Payment Cryptography Cryptographic Details](#).

Infrastructure security in AWS Payment Cryptography

As a managed service, AWS Payment Cryptography is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access AWS Payment Cryptography through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Isolation of physical hosts

The security of the physical infrastructure that AWS Payment Cryptography uses is subject to the controls described in the Physical and Environmental Security section of [Amazon Web Services: Overview of Security Processes](#). You can find more detail in compliance reports and third-party audit findings listed in the previous section.

AWS Payment Cryptography is supported by dedicated commercial-off-the-shelf PCI PTS HSM-listed hardware security modules (HSMs). The key material for AWS Payment Cryptography keys is stored only in volatile memory on the HSMs, and only while the Payment Cryptography key is in use. HSMs are in access controlled racks within Amazon data centers that enforce dual control for any physical access. For detailed information about the operation of AWS Payment Cryptography HSMs, see [AWS Payment Cryptography Cryptographic Details](#).

Connecting to AWS Payment Cryptography through a VPC endpoint

You can connect directly to AWS Payment Cryptography through a private interface endpoint in your virtual private cloud (VPC). When you use an interface VPC endpoint, communication between your VPC and AWS Payment Cryptography is conducted entirely within the AWS network.

AWS Payment Cryptography supports Amazon Virtual Private Cloud (Amazon VPC) endpoints powered by [AWS PrivateLink](#). Each VPC endpoint is represented by one or more [Elastic Network Interfaces](#) (ENIs) with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to AWS Payment Cryptography without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC do not need public IP addresses to communicate with AWS Payment Cryptography.

Regions

AWS Payment Cryptography supports VPC endpoints and VPC endpoint policies in all AWS Regions in which [AWS Payment Cryptography](#) is supported.

Topics

- [Considerations for AWS Payment Cryptography VPC endpoints](#)
- [Creating a VPC endpoint for AWS Payment Cryptography](#)
- [Connecting to an AWS Payment Cryptography VPC endpoint](#)
- [Controlling access to a VPC endpoint](#)
- [Using a VPC endpoint in a policy statement](#)
- [Logging your VPC endpoint](#)

Considerations for AWS Payment Cryptography VPC endpoints

Note

Although VPC endpoints allows you to connect to the service in as few as one availability zone(AZ), we recommend connecting to three availability zones for high availability and redundancy purposes.

Before you set up an interface VPC endpoint for AWS Payment Cryptography, review the [Interface endpoint properties and limitations](#) topic in the *AWS PrivateLink Guide*.

AWS Payment Cryptography support for a VPC endpoint includes the following.

- You can use your VPC endpoint to call all [AWS Payment Cryptography Controlplane operations](#) and [AWS Payment Cryptography Dataplane operations](#) from a VPC.
- You can create an interface VPC endpoint that connects to an AWS Payment Cryptography region endpoint.
- AWS Payment Cryptography consists of a control plane and a data plane. You can chose to setup one or both sub-services but each is configured separately.
- You can use AWS CloudTrail logs to audit your use of AWS Payment Cryptography keys through the VPC endpoint. For details, see [Logging your VPC endpoint](#).

Creating a VPC endpoint for AWS Payment Cryptography

You can create a VPC endpoint for AWS Payment Cryptography by using the Amazon VPC console or the Amazon VPC API. For more information, see [Create an interface endpoint](#) in the *AWS PrivateLink Guide*.

- To create a VPC endpoint for AWS Payment Cryptography, use the following service names:

```
com.amazonaws.region.payment-cryptography.controlplane
```

```
com.amazonaws.region.payment-cryptography.dataplane
```

For example, in the US West (Oregon) Region (us-west-2), the service names would be:

```
com.amazonaws.us-west-2.payment-cryptography.controlplane
```

```
com.amazonaws.us-west-2.payment-cryptography.dataplane
```

To make it easier to use the VPC endpoint, you can enable a [private DNS name](#) for your VPC endpoint. If you select the **Enable DNS Name** option, the standard AWS Payment Cryptography DNS hostname resolves to your VPC endpoint. For example, `https://controlplane.payment-`

cryptography.us-west-2.amazonaws.com would resolve to a VPC endpoint connected to service name com.amazonaws.us-west-2.payment-cryptography.controlplane.

This option makes it easier to use the VPC endpoint. The AWS SDKs and AWS CLI use the standard AWS Payment Cryptography DNS hostname by default, so you do not need to specify the VPC endpoint URL in applications and commands.

For more information, see [Accessing a service through an interface endpoint](#) in the *AWS PrivateLink Guide*.

Connecting to an AWS Payment Cryptography VPC endpoint

You can connect to AWS Payment Cryptography through the VPC endpoint by using an AWS SDK, the AWS CLI or AWS Tools for PowerShell. To specify the VPC endpoint, use its DNS name.

For example, this [list-keys](#) command uses the `endpoint-url` parameter to specify the VPC endpoint. To use a command like this, replace the example VPC endpoint ID with one in your account.

```
$ aws payment-cryptography list-keys --endpoint-url https://  
vpce-1234abcdef5678c90a-09p7654s-us-east-1a.ec2.us-east-1.vpce.amazonaws.com
```

If you enabled private hostnames when you created your VPC endpoint, you do not need to specify the VPC endpoint URL in your CLI commands or application configuration. The standard AWS Payment Cryptography DNS hostname resolves to your VPC endpoint. The AWS CLI and SDKs use this hostname by default, so you can begin using the VPC endpoint to connect to an AWS Payment Cryptography regional endpoint without changing anything in your scripts and applications.

To use private hostnames, the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC must be set to `true`. To set these attributes, use the [ModifyVpcAttribute](#) operation. For details, see [View and update DNS attributes for your VPC](#) in the *Amazon VPC User Guide*.

Controlling access to a VPC endpoint

To control access to your VPC endpoint for AWS Payment Cryptography, attach a *VPC endpoint policy* to your VPC endpoint. The endpoint policy determines whether principals can use the VPC endpoint to call AWS Payment Cryptography operations with specific AWS Payment Cryptography resources.

You can create a VPC endpoint policy when you create your endpoint, and you can change the VPC endpoint policy at any time. Use the VPC management console, or the [CreateVpcEndpoint](#) or [ModifyVpcEndpoint](#) operations. You can also create and change a VPC endpoint policy by [using an AWS CloudFormation template](#). For help using the VPC management console, see [Create an interface endpoint](#) and [Modifying an interface endpoint](#) in the *AWS PrivateLink Guide*.

Topics

- [About VPC endpoint policies](#)
- [Default VPC endpoint policy](#)
- [Creating a VPC endpoint policy](#)
- [Viewing a VPC endpoint policy](#)

About VPC endpoint policies

For an AWS Payment Cryptography request that uses a VPC endpoint to be successful, the principal requires permissions from two sources:

- An [identity-based policy](#) must give the principal permission to call the operation on the resource (AWS Payment Cryptography keys or alias).
- A VPC endpoint policy must give the principal permission to use the endpoint to make the request.

For example, a key policy might give a principal permission to call [Decrypt](#) on a particular AWS Payment Cryptography keys. However, the VPC endpoint policy might not allow that principal to call `Decrypt` on that AWS Payment Cryptography keys by using the endpoint.

Or a VPC endpoint policy might allow a principal to use the endpoint to call [StopKeyUsage](#) on certain AWS Payment Cryptography keys. But if the principal doesn't have those permissions from a IAM policy, the request fails.

Default VPC endpoint policy

Every VPC endpoint has a VPC endpoint policy, but you are not required to specify the policy. If you don't specify a policy, the default endpoint policy allows all operations by all principals on all resources over the endpoint.

However, for AWS Payment Cryptography resources, the principal must also have permission to call the operation from an [IAM policy](#). Therefore, in practice, the default policy says that if a principal has permission to call an operation on a resource, they can also call it by using the endpoint.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

To allow principals to use the VPC endpoint for only a subset of their permitted operations, [create or update the VPC endpoint policy](#).

Creating a VPC endpoint policy

A VPC endpoint policy determines whether a principal has permission to use the VPC endpoint to perform operations on a resource. For AWS Payment Cryptography resources, the principal must also have permission to perform the operations from an [IAM policy](#).

Each VPC endpoint policy statement requires the following elements:

- The principal that can perform actions
- The actions that can be performed
- The resources on which actions can be performed

The policy statement doesn't specify the VPC endpoint. Instead, it applies to any VPC endpoint to which the policy is attached. For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of a VPC endpoint policy for AWS Payment Cryptography. When attached to a VPC endpoint, this policy allows `ExampleUser` to use the VPC endpoint to call the specified operations on the specified AWS Payment Cryptography keys. Before using a policy like this one, replace the example principal and [key identifier](#) with valid values from your account.

```
{
```



```

"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 logs all operations that use the VPC endpoint. However, your CloudTrail logs don't include operations requested by principals in other accounts or operations for AWS Payment Cryptography keys in other accounts.

As such, you might want to create a VPC endpoint policy that prevents principals in external accounts from using the VPC endpoint to call any AWS Payment Cryptography operations on any keys in the local account.

The following example uses the [aws:PrincipalAccount](#) global condition key to deny access to all principals for all operations on all AWS Payment Cryptography keys unless the principal is in the local account. Before using a policy like this one, replace the example account ID with a valid one.

```

{
  "Statement": [
    {
      "Sid": "AccessForASpecificAccount",
      "Principal": {"AWS": "*"},
      "Action": "payment-cryptography:*",
      "Effect": "Deny",
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalAccount": "111122223333"
        }
      }
    }
  ]
}

```

```
    }  
  }  
]  
}
```

Viewing a VPC endpoint policy

To view the VPC endpoint policy for an endpoint, use the [VPC management console](#) or the [DescribeVpcEndpoints](#) operation.

The following AWS CLI command gets the policy for the endpoint with the specified VPC endpoint ID.

Before using this command, replace the example endpoint ID with a valid one from your account.

```
$ aws ec2 describe-vpc-endpoints \  
--query 'VpcEndpoints[?VpcEndpointId==`vpce-1234abcdef5678c90a`].[PolicyDocument]'  
--output text
```

Using a VPC endpoint in a policy statement

You can control access to AWS Payment Cryptography resources and operations when the request comes from VPC or uses a VPC endpoint. To do so, use one an [IAM policy](#)

- Use the `aws:sourceVpce` condition key to grant or restrict access based on the VPC endpoint.
- Use the `aws:sourceVpc` condition key to grant or restrict access based on the VPC that hosts the private endpoint.

Note

The `aws:sourceIP` condition key is not effective when the request comes from an [Amazon VPC endpoint](#). To restrict requests to a VPC endpoint, use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see [Identity and access management for VPC endpoints and VPC endpoint services](#) in the *AWS PrivateLink Guide*.

You can use these global condition keys to control access to AWS Payment Cryptography keys, aliases, and to operations like [CreateKey](#) that don't depend on any particular resource.

For example, the following sample key policy allows a user to perform particular cryptographic operations with a AWS Payment Cryptography keys only when the request uses the specified VPC endpoint, blocking access both from the Internet and connections (if setup). When a user makes a request to AWS Payment Cryptography, the VPC endpoint ID in the request is compared to the `aws:sourceVpce` condition key value in the policy. If they do not match, the request is denied.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
  "Id": "example-key-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable IAM policies",
      "Effect": "Allow",
      "Principal": {"AWS":["111122223333"]},
      "Action": ["payment-cryptography:*"],
      "Resource": "*"
    },
    {
      "Sid": "Restrict usage to my VPC endpoint",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "payment-cryptography:Encrypt",
        "payment-cryptography:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:sourceVpce": "vpce-1234abcd5678c90a"
        }
      }
    }
  ]
}
```

You can also use the `aws:sourceVpc` condition key to restrict access to your AWS Payment Cryptography keys based on the VPC in which VPC endpoint resides.

The following sample key policy allows commands that manage the AWS Payment Cryptography keys only when they come from `vpc-12345678`. In addition, it allows commands that use the AWS Payment Cryptography keys for cryptographic operations only when they come from `vpc-2b2b2b2b`. You might use a policy like this one if an application is running in one VPC, but you use a second, isolated VPC for management functions.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
  "Id": "example-key-2",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow administrative actions from vpc-12345678",
      "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": "Allow key usage from vpc-2b2b2b2b",
      "Effect": "Allow",
      "Principal": {"AWS": "111122223333"},
      "Action": [
        "payment-cryptography:Encrypt", "payment-cryptography:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:sourceVpc": "vpc-2b2b2b2b"
        }
      }
    }
  ]
}
```

```

    }
  },
  {
    "Sid": "Allow list/read actions from everywhere",
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": [
      "payment-cryptography:List*", "payment-cryptography:Get*"
    ],
    "Resource": "*",
  }
]
}

```

Logging your VPC endpoint

AWS CloudTrail logs all operations that use the VPC endpoint. When a request to AWS Payment Cryptography uses a VPC endpoint, the VPC endpoint ID appears in the [AWS CloudTrail log](#) entry that records the request. You can use the endpoint ID to audit the use of your AWS Payment Cryptography VPC endpoint.

To protect your VPC, requests that are denied by a [VPC endpoint policy](#), but otherwise would have been allowed, are not recorded in [AWS CloudTrail](#).

For example, this sample log entry records a [GenerateMac](#) request that used the VPC endpoint. The `vpcEndpointId` field appears at the end of the log entry.

```

{
  "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/
kwapwa6qaiifllw2h",
    "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",
"vpceEndpointId": "vpce-1234abcd5678c90a",
"eventCategory": "Management",
"tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_128_GCM_SHA256",
    "clientProvidedHostHeader": "vpce-1234abcd5678c90a-
oo28vrivr.controlplane.payment-cryptography.us-east-1.vpce.amazonaws.com"
}
}

```

Security best practices for AWS Payment Cryptography

AWS Payment Cryptography supports many security features that are either built-in or that you can optionally implement to enhance the protection of your encryption keys and ensure that they are used for their intended purpose, including [IAM policies](#), an extensive set of policy condition

keys to refine your key policies and IAM policies and built-in enforcement of PCI PIN rules regarding key blocks.

Important

The general guidelines provided do not represent a complete security solution. Because not all best practices are appropriate for all situations, these are not intended to be prescriptive.

- **Key Usage and Modes of Use:** AWS Payment Cryptography follows and enforces key usage and mode of use restrictions as described in ANSI X9 TR 31-2018 Interoperable Secure Key Exchange Key Block Specification and consistent with PCI PIN Security Requirement 18-3. This limits the ability to use a single key for multiple purposes and cryptographically binds the key metadata (such as permitted operations) to the key material itself. AWS Payment Cryptography automatically enforces these restrictions such as that a key encryption key (TR31_KO_KEY_ENCRYPTION_KEY) cannot also be used for data decryption. See [Understanding key attributes for AWS Payment Cryptography key](#) for more details.
- **Limit sharing of symmetric key material:** Only share symmetric key material (such as Pin Encryption Keys or Key Encryption Keys) with at most one other entity. If there is a need to transit sensitive material to more entities or partners, create additional keys. AWS Payment Cryptography never exposes symmetric key material or asymmetric private key material in the clear.
- **Use aliases or tags to associate keys with certain use cases or partners:** Aliases can be used to easily denote the use case associated with a key such as alias/BIN_12345_CVK to denote a card verification key associated with BIN 12345. To provide more flexibility, consider creating tags such as bin=12345, use_case=acquiring,country=us,partner=foo. Aliases and tags can also be used to limit access such as enforcing access controls between issuing and acquiring use cases.
- **Practice least privileged access:** IAM can be used to limit production access to systems rather than individuals, such as prohibiting individual users from creating keys or running cryptographic operations. IAM can also be used to limit access to both commands and keys that may not be applicable for your use case, such as limiting the ability to generate or validate pins for an acquirer. Another way to use least privileged access is to restrict sensitive operations (such as key import) to specific service accounts. See [AWS Payment Cryptography identity-based policy examples](#) for examples.

See also

- [Identity and access management for AWS Payment Cryptography](#)
- [Security best practices in IAM](#) in the *IAM User Guide*

Compliance validation for AWS Payment Cryptography

Third-party auditors assess the security and compliance of AWS Payment Cryptography as part of multiple AWS compliance programs. These include SOC, PCI, and others.

AWS Payment Cryptography has been assessed for several PCI standards in addition to PCI DSS. These include PCI PIN Security (PCI PIN) and PCI Point-to-Point (P2PE) Encryption. Please see [AWS Artifact](#) for available attestations and compliance guides.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS Payment Cryptography is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#)—These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [AWS Compliance Resources](#)—This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide*—AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#)—This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Identity and access management for AWS Payment Cryptography

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS Payment Cryptography resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS Payment Cryptography works with IAM](#)
- [AWS Payment Cryptography identity-based policy examples](#)
- [Troubleshooting AWS Payment Cryptography identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS Payment Cryptography.

Service user – If you use the AWS Payment Cryptography service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Payment Cryptography features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS Payment Cryptography, see [Troubleshooting AWS Payment Cryptography identity and access](#).

Service administrator – If you're in charge of AWS Payment Cryptography resources at your company, you probably have full access to AWS Payment Cryptography. It's your job to determine which AWS Payment Cryptography features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS Payment Cryptography, see [How AWS Payment Cryptography works with IAM](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Payment Cryptography. To view example AWS Payment Cryptography identity-based policies that you can use in IAM, see [AWS Payment Cryptography identity-based policy examples](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [AWS Multi-factor authentication in IAM](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account.

We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can [switch from a user to an IAM role \(console\)](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set.

To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
 - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
 - **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
 - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that

are running on the EC2 instance to get temporary credentials. For more information, see [Use an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If

you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [Service control policies](#) in the *AWS Organizations User Guide*.

- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see [Resource control policies \(RCPs\)](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How AWS Payment Cryptography works with IAM

Before you use IAM to manage access to AWS Payment Cryptography, you should understand what IAM features are available to use with AWS Payment Cryptography. To get a high-level view of how AWS Payment Cryptography and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

Topics

- [AWS Payment Cryptography Identity-based policies](#)
- [Authorization based on AWS Payment Cryptography tags](#)

AWS Payment Cryptography Identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. AWS Payment Cryptography supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in AWS Payment Cryptography use the following prefix before the action: `payment-cryptography:.` For example, to grant someone permission to execute an AWS Payment Cryptography `VerifyCardData` API operation, you include the `payment-cryptography:VerifyCardData` action in their policy. Policy statements must include either an **Action** or **NotAction** element. AWS Payment Cryptography defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "payment-cryptography:action1",  
    "payment-cryptography:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List` (such as `ListKeys` and `ListAliases`), include the following action:

```
"Action": "payment-cryptography:List*"
```

To see a list of AWS Payment Cryptography actions, see [Actions Defined by AWS Payment Cryptography](#) in the *IAM User Guide*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The payment-cryptography key resource has the following ARN:

```
arn:${Partition}:payment-cryptography:${Region}:${Account}:key/${keyARN}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#).

For example, to specify the `arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qai11w2h` instance in your statement, use the following ARN:

```
"Resource": "arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qai11w2h"
```

To specify all keys that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:payment-cryptography:us-east-2:111122223333:key/*"
```

Some AWS Payment Cryptography actions, such as those for creating keys, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To specify multiple resources in a single statement, use a comma as shown below:

```
"Resource": [  
    "resource1",  
    "resource2"
```

Examples

To view examples of AWS Payment Cryptography identity-based policies, see [AWS Payment Cryptography identity-based policy examples](#).

Authorization based on AWS Payment Cryptography tags

AWS Payment Cryptography identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS Payment Cryptography resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the *IAM User Guide*.

Topics

- [Policy best practices](#)
- [Using the AWS Payment Cryptography console](#)
- [Allow users to view their own permissions](#)
- [Ability to access all aspects of AWS Payment Cryptography](#)
- [Ability to call APIs using specified keys](#)
- [Ability to specifically deny a resource](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Payment Cryptography resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the AWS Payment Cryptography console

To access the AWS Payment Cryptography console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS Payment Cryptography resources in your AWS account. If you create an identity-based policy that is more restrictive than

the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the AWS Payment Cryptography console, also attach the following AWS managed policy to the entities. For more information, see [Adding Permissions to a User](#) in the *IAM User Guide*.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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": "*"
}
]
}

```

Ability to access all aspects of AWS Payment Cryptography

Warning

This example provides wide permissions and is not recommended. Consider least privileged access models instead.

In this example, you want to grant an IAM user in your AWS account access to all of your AWS Payment Cryptography keys and the ability to call all AWS Payment Cryptography apis including both ControlPlane and DataPlane operations.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "payment-cryptography:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

Ability to call APIs using specified keys

In this example, you want to grant an IAM user in your AWS account access to one of your AWS Payment Cryptography key, `arn:aws:payment-cryptography:us-`

east-2:111122223333:key/kwapwa6qaiif1lw2h and then use this resource in two APIs, `GenerateCardData` and `VerifyCardData`. Conversely, the IAM user will not have access to use this key on other operations such as `DeleteKey` or `ExportKey`

Resources can be either keys, prefixed with `key` or aliases, prefixed with `alias`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "payment-cryptography:VerifyCardData",
        "payment-cryptography:GenerateCardData"
      ],
      "Resource": [
        "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaiif1lw2h"
      ]
    }
  ]
}
```

Ability to specifically deny a resource

Warning

Carefully consider the implications of granting wildcard access. Consider a least privilege model instead.

In this example, you want to permit an IAM user in your AWS account access to any of your AWS Payment Cryptography key but want to deny permissions to one specific key. The user will have access to `VerifyCardData` and `GenerateCardData` with all keys with the exception of the one specified in the deny statement.

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



```
{
  "Effect": "Allow",
  "Action": [
    "payment-cryptography:VerifyCardData",
    "payment-cryptography:GenerateCardData"
  ],
  "Resource": [
    "arn:aws:payment-cryptography:us-east-2:111122223333:key/*"
  ]
},
{
  "Effect": "Deny",
  "Action": [
    "payment-cryptography:GenerateCardData"
  ],
  "Resource": [
    "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaiifllw2h"
  ]
}
]
```

Troubleshooting AWS Payment Cryptography identity and access

Topics will be added to this section as IAM-related issues that are specific to AWS Payment Cryptography are identified. For general troubleshooting content on IAM topics, refer to the [troubleshooting section](#) of the *IAM User Guide*.

Monitoring AWS Payment Cryptography

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Payment Cryptography and your other AWS solutions. AWS provides the following monitoring tools to watch AWS Payment Cryptography, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track usage of certain APIs or notify you if you are approaching your AWS Payment Cryptography quotas. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the endpoint called, the resources(keys) used, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).

Topics

- [Logging AWS Payment Cryptography API calls using AWS CloudTrail](#)

Logging AWS Payment Cryptography API calls using AWS CloudTrail

AWS Payment Cryptography is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Payment Cryptography. CloudTrail captures all API calls for AWS Payment Cryptography as events. The calls captured include calls from the console and code calls to the API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Payment

Cryptography. If you don't configure a trail, you can still view the most recent management (Control Plane) events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS Payment Cryptography, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Topics

- [AWS Payment Cryptography information in CloudTrail](#)
- [Control plane events in CloudTrail](#)
- [Data events in CloudTrail](#)
- [Understanding AWS Payment Cryptography Control Plane log file entries](#)
- [Understanding AWS Payment Cryptography Data Plane log file entries](#)

AWS Payment Cryptography information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Payment Cryptography, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for AWS Payment Cryptography, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple Regions](#)
- [Receiving CloudTrail log files from multiple accounts](#)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity element](#).

Control plane events in CloudTrail

CloudTrail logs AWS Payment Cryptography operations, such as [CreateKey](#), [ImportKey](#), [DeleteKey](#), [ListKeys](#), [TagResource](#), and all other control plane operations.

Data events in CloudTrail

[Data events](#) provide information about the resource operations performed on or in a resource, such as encrypting a payload or translating a pin. Data events are high-volume activities that CloudTrail does not log by default. You can enable data events API action logging for AWS Payment Cryptography dataplane events by using CloudTrail APIs or console. For more information, see [Logging data events](#) in the *AWS CloudTrail User Guide*.

With CloudTrail, you must use advanced event selectors to decide which AWS Payment Cryptography API activities are logged and recorded. To log AWS Payment Cryptography dataplane events, you must include the resource type `AWS Payment Cryptography key` and `AWS Payment Cryptography alias`. Once this is set, you can refine your logging preferences further by selecting specific data events for recording, such as using the `eventName` filter to track `EncryptData` events. For more information, see [AdvancedEventSelector](#) in the *AWS CloudTrail API Reference*.

Note

To subscribe to AWS Payment Cryptography data events, you must utilize advanced event selectors. We recommend subscribing to key and alias events to ensure that you receive all events.

data events:

- [DecryptData](#)
- [EncryptData](#)
- [GenerateCardValidationData](#)
- [GenerateMac](#)
- [GeneratePinData](#)
- [ReEncryptData](#)
- [TranslatePinData](#)
- [VerifyAuthRequestCryptogram](#)
- [VerifyCardValidationData](#)
- [VerifyMac](#)
- [VerifyPinData](#)

Additional charges apply for data events. For more information, see [AWS CloudTrail Pricing](#).

Understanding AWS Payment Cryptography Control Plane log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the AWS Payment Cryptography CreateKey action.

```
{
  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/5rplquuwozodpwp,
      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=: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={
  }
}
}
}

```

Understanding AWS Payment Cryptography Data Plane log file entries

Dataplane events can optionally be configured and function similarly to Controlplane logs but are typically much higher volumes. Given the sensitive nature of some inputs and outputs to AWS Payment Cryptography dataplane operations, you may find certain fields with the message "*** Sensitive Data Redacted ***". This is not configurable and is intended to prevent sensitive data from appearing in logs or trails.

The following example shows a CloudTrail log entry that demonstrates the AWS Payment Cryptography EncryptData action.

```

{
  "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",

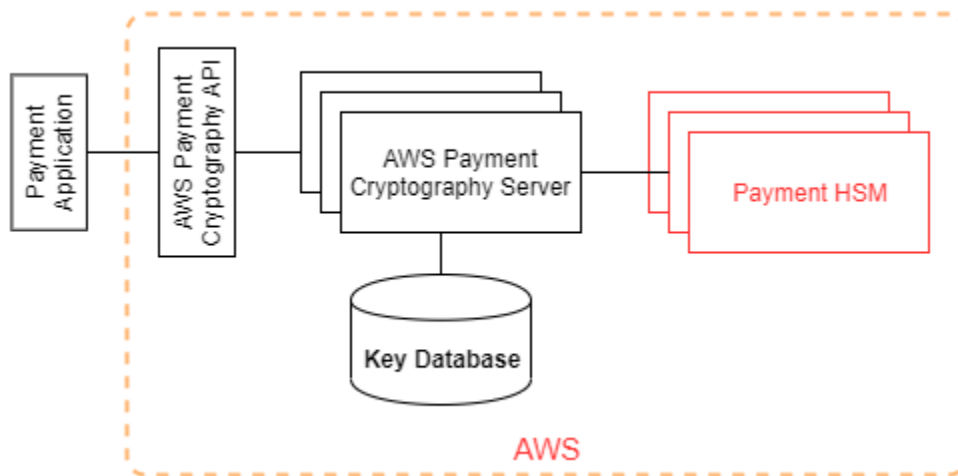
```

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

Cryptographic details

AWS Payment Cryptography provides a web interface to generate and manage cryptographic keys for payment transactions. AWS Payment Cryptography offers standard key management services and payment transaction cryptography and tools you can use for centralized management and auditing. This documentation provides a detailed description of the cryptographic operations you can use in AWS Payment Cryptography to assist you in evaluating the features offered by the service.

AWS Payment Cryptography contains multiple interfaces (including a RESTful API, through the AWS CLI, AWS SDK and the AWS Management Console) to request cryptographic operations of a distributed fleet of [PCI PTS HSM-validated hardware security modules](#).



AWS Payment Cryptography is a tiered service consisting of web-facing AWS Payment Cryptography hosts and a tier of HSMs. The grouping of these tiered hosts forms the AWS Payment Cryptography stack. All requests to AWS Payment Cryptography must be made over the Transport Layer Security protocol (TLS) and terminate on an AWS Payment Cryptography host. The service hosts only allow TLS with a cipher suite that provides [perfect forward secrecy](#). The service authenticates and authorizes your requests using the same credential and policy mechanisms of IAM that are available for all other AWS API operations.

AWS Payment Cryptography servers connect to the underlying [HSM](#) via a private, non-virtual network. Connections between service components and [HSM](#) are secured with mutual TLS (mTLS) for authentication and encryption.

Design goals

AWS Payment Cryptography is designed to meet the following requirements:

- **Trustworthy** — Use of keys is protected by access control policies that you define and manage. There is no mechanism to export plaintext AWS Payment Cryptography keys. The confidentiality of your cryptographic keys is crucial. Multiple Amazon employees with role-specific access to quorum-based access controls are required to perform administrative actions on the HSMs. No Amazon employees have access to HSM main (or master) keys or backups. Main keys cannot be synchronized with HSMs that are not part of an AWS Payment Cryptography region. All other keys are protected by HSM main keys. Therefore, customer AWS Payment Cryptography keys are not usable outside of the AWS Payment Cryptography service operating within a customer's account.
- **Low-latency and high throughput** — AWS Payment Cryptography provides cryptographic operations at latency and throughput level suitable for managing payment cryptographic keys and processing payment transactions.
- **Durability** — The durability of cryptographic keys is designed to be equal that of the highest durability services in AWS. A single cryptographic key can be shared with a payment terminal, EMV chip card, or other secure cryptographic device (SCD) that is in use for many years.
- **Independent Regions** — AWS provides independent regions for customers who need to restrict data access in different regions or need to comply with data residency requirements. Key usage can be isolated within an AWS Region.
- **Secure source of random numbers** — Because strong cryptography depends on truly unpredictable random number generation, AWS Payment Cryptography provides a high-quality and validated source of random numbers. All key generation for AWS Payment Cryptography uses PCI PTS HSM-listed HSM, operating in PCI mode.
- **Audit** — AWS Payment Cryptography records the use and management of cryptographic keys in CloudTrail logs and service logs available via Amazon CloudWatch. You can use CloudTrail logs to inspect use of your cryptographic keys, including the use of keys by accounts that you have shared keys with. AWS Payment Cryptography is audited by third party assessors against applicable PCI, card brand, and regional payment security standards. Attestations and Shared Responsibility guides are available on AWS Artifact.
- **Elastic** — AWS Payment Cryptography scales out and in according to your demand. Instead of predicting and reserving HSM capacity, AWS Payment Cryptography provides payment cryptography on-demand. AWS Payment Cryptography takes responsibility for maintaining the security and compliance of HSM to provide sufficient capacity to meet customer's peak demand.

Foundations

The topics in this chapter describe the cryptographic primitives of AWS Payment Cryptography and where they are used. They also introduce the basic elements of the service.

Topics

- [Cryptographic primitives](#)
- [Entropy and random number generation](#)
- [Symmetric key operations](#)
- [Asymmetric key operations](#)
- [Key storage](#)
- [Key import using symmetric keys](#)
- [Key import using asymmetric keys](#)
- [Key export](#)
- [Derived Unique Key Per Transaction \(DUKPT\) protocol](#)
- [Key hierarchy](#)

Cryptographic primitives

AWS Payment Cryptography uses parameter-able, standard cryptographic algorithms so that applications can implement the algorithms needed for their use case. The set of cryptographic algorithms is defined by PCI, ANSI X9, EMVco, and ISO standards. All cryptography is performed by PCI PTS HSM standard-listed HSMs running in PCI mode.

Entropy and random number generation

AWS Payment Cryptography key generation is performed on the AWS Payment Cryptography HSMs. The HSMs implement a random number generator that meets the PCI PTS HSM requirement for all supported key types and parameters.

Symmetric key operations

Symmetric key algorithms and key strengths defined in ANSI X9 TR 31, ANSI X9.24, and PCI PIN Annex C are supported:

- **Hash functions** — Algorithms from the SHA2 and SHA3 family with output size greater than 2551. Except for backwards compatibility with pre-PCI PTS POI v3 terminals.
- **Encryption and decryption** — AES with key size greater than or equal to 128 bits, or TDEA with keys size greater than or equal to 112 bits (2 key or 3 key).
- **Message Authentication Codes (MACs)** CMAC or GMAC with AES, as well as HMAC with an approved hash function and a key size greater than or equal to 128.

AWS Payment Cryptography uses AES 256 for HSM main keys, data protection keys, and TLS session keys.

Note: Some of the listed functions are used internally to support standard protocols and data structures. See the API documentation for algorithms supported by specific actions.

Asymmetric key operations

Asymmetric key algorithms and key strengths defined in ANSI X9 TR 31, ANSI X9.24, and PCI PIN Annex C are supported:

- **Approved key establishment schemes** — as described in NIST SP800-56A (ECC/FCC2-based key agreement), NIST SP800-56B (IFC-based key agreement), and NIST SP800-38F (AES-based key encryption/wrapping).

AWS Payment Cryptography hosts only allow connections to the service using TLS with a cipher suite that provides [perfect forward secrecy](#).

Note: Some of the listed functions are used internally to support standard protocols and data structures. See the API documentation for algorithms supported by specific actions.

Key storage

AWS Payment Cryptography keys are protected by HSM AES 256 main keys and stored in ANSI X9 TR 31 key blocks in an encrypted database. The database is replicated to in-memory database on AWS Payment Cryptography servers.

According to PCI PIN Security Normative Annex C, AES 256 keys are equally as strong as or stronger than:

- 3-key TDEA

- RSA 15360 bit
- ECC 512 bit
- DSA, DH, and MQV 15360/512

Key import using symmetric keys

AWS Payment Cryptography supports import of cryptograms and key blocks with symmetric or public keys with a symmetric key encryption key (KEK) that is as strong or stronger than the protected key for import.

Key import using asymmetric keys

AWS Payment Cryptography supports import of cryptograms and key blocks with symmetric or public keys protected by a private key encryption key (KEK) that is as strong or stronger than the protected key for import. The public key provided for decryption must have its authenticity and integrity ensured by a certificate from an authority trusted by the customer.

Public KEK provided by AWS Payment Cryptography have the authentication and integrity protection of a certificate authority (CA) with attested compliance to PCI PIN Security and PCI P2PE Annex A.

Key export

Keys can be exported and protected by keys with the appropriate KeyUsage and that are as strong as or stronger than the key to be exported.

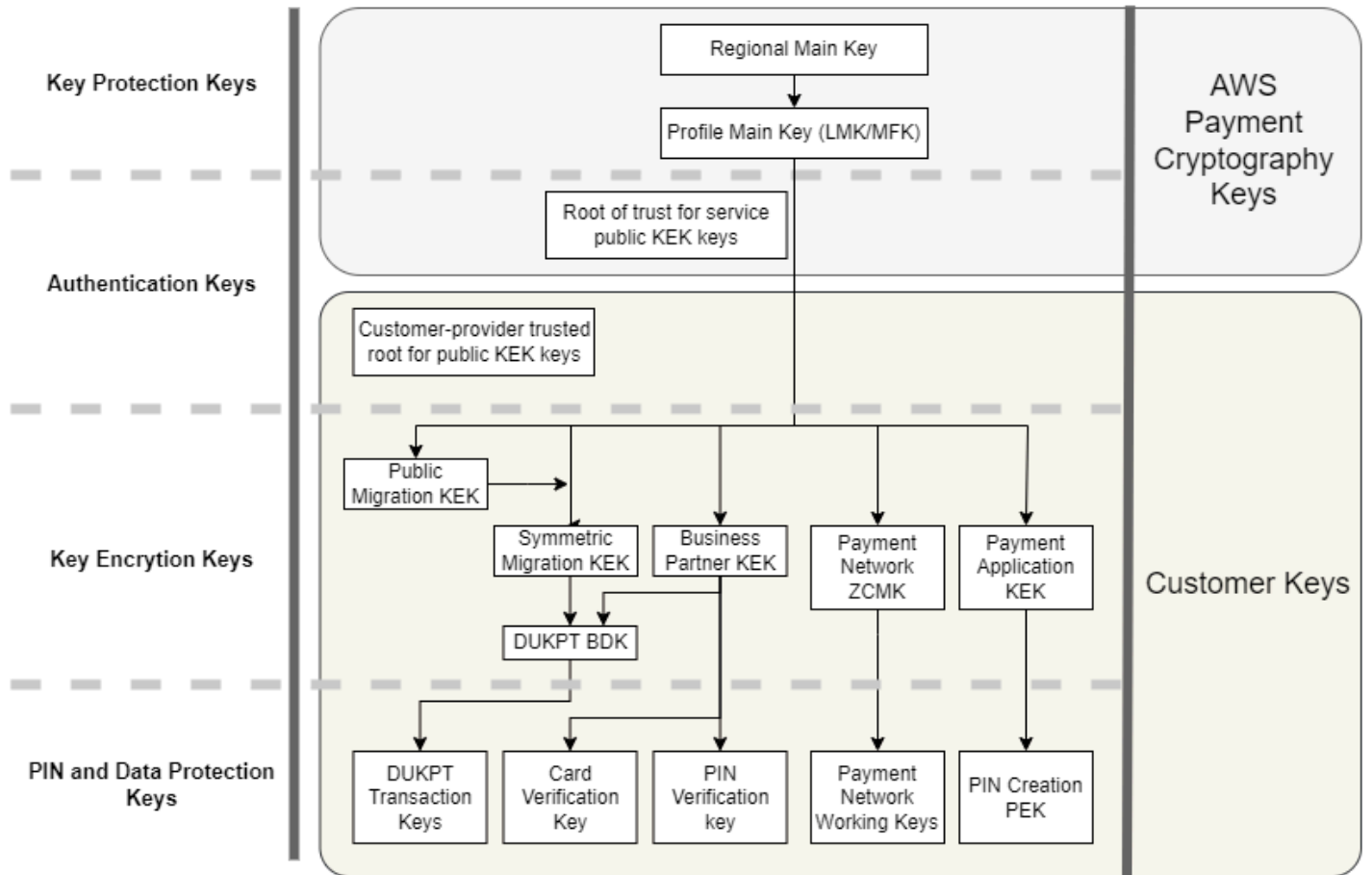
Derived Unique Key Per Transaction (DUKPT) protocol

AWS Payment Cryptography supports with TDEA and AES base derivation keys (BDK) as described by ANSI X9.24-3.

Key hierarchy

The AWS Payment Cryptography key hierarchy ensures that keys are always protected by keys as strong as or stronger than the keys they protect.

Payment Cryptographic Keys



AWS Payment Cryptography keys are used for key protection within the service:

Key	Description
Regional Main Key	Protects virtual HSM images, or profiles, used for cryptographic processing. This key exists only in HSM and secure backups.
Profile Main Key	Top level customer key protection key, traditionally called a Local Master Key (LMK) or Master File Key (MFK) for customer keys. This key exists only in HSM and secure backups. Profiles define distinct HSM configurations as required by security standards for payments use cases.

Key	Description
Root of trust for AWS Payment Cryptography public key encryption key (KEK) keys	The trusted root public key and certificate for authenticating and validating public keys supplied by AWS Payment Cryptography for key import and export using asymmetric keys.

Customer keys are grouped by keys used to protect other keys and keys that protect payment-related data. These are examples of customer keys of both types:

Key	Description
Customer-provided trusted root for public KEK keys	Public key and certificate supplied by you as the root of trust for authenticating and validating public keys that you supply for key import and export using asymmetric keys.
Key Encryption Keys (KEK)	KEK are used solely to encrypt other keys for exchange between external key stores and AWS Payment Cryptography, business partners, payment networks, or different applications within your organization.
Derived Unique Key Per Transaction (DUKPT) base derivation key (BDK)	BDKs are used to create unique keys for each payment terminal and translate transactions from multiple terminals to a single acquiring bank, or acquirer, working key. The best practice, which is required by PCI Point-to-Point Encryption (P2PE), is that different BDKs are used for different terminal models, key injection or initialization services, or other segmentation to limit the impact of compromising a BDK.

Key	Description
Payment network zone control master key (ZCMK)	ZCMK, also referred to as zone keys or zone master keys, are provided by payment networks to establish initial working keys.
DUKPT transaction keys	Payment terminals configured for DUKPT derive a unique key for the terminal and transaction. The HSM receiving the transaction can determine the key from the terminal identifier and transaction sequence number.
Card data preparation keys	EMV issuer master keys, EMV card keys and verification values, and card personalization data file protection keys are used to create data for individual cards for use by a card personalization provider. These keys and cryptographic validation data are also used by issuing banks, or issuers, for authenticating card data as part of authorizing transactions.
Card data preparation keys	EMV issuer master keys, EMV card keys and verification values, and card personalization data file protection keys are used to create data for individual cards for use by a card personalization provider. These keys and cryptographic validation data are also used by issuing banks, or issuers, for authenticating card data as part of authorizing transactions.
Payment network working keys	Often referred to as issuer working key or acquirer working key, these are the keys that encrypt transaction sent to or received from payment networks. These keys are rotated frequently by the network, often daily or hourly. These are PIN encryption keys (PEK) for PIN/Debit transactions.

Key	Description
Personal Identification Number (PIN) encryption keys (PEK)	Applications that create or decrypt PIN blocks use PEK to prevent storage or transmission of clear text PIN.

Internal operations

This topic describes internal requirements implemented by the service to secure customer keys and cryptographic operations for a globally distributed and scalable payment cryptography and key management service.

Topics

- [HSM specifications and lifecycle](#)
- [HSM device physical security](#)
- [HSM initialization](#)
- [HSM service and repair](#)
- [HSM decommissioning](#)
- [HSM firmware update](#)
- [Operator access](#)
- [Key management](#)

HSM specifications and lifecycle

AWS Payment Cryptography uses a fleet of commercially available HSM. The HSMs are FIPS 140-2 Level 3 validated and also use firmware versions and the security policy listed on the PCI Security Standards Council [approved PCI PTS Devices list](#) as PCI HSM v3 complaint. The PCI PTS HSM standard includes additional requirements for the manufacturing, shipment, deployment, management, and destruction of HSM hardware which are important for payment security and compliance but not addressed by FIPS 140.

All HSMs are operated in PCI Mode and configured with the PCI PTS HSM security policy. Only functions required to support AWS Payment Cryptography use cases are enabled. AWS Payment Cryptography does not provide for printing, display, or return of clear text PINs.

HSM device physical security

Only HSMs that have device keys signed by an AWS Payment Cryptography certificate authority (CA) by the manufacturer prior to delivery can be used by the service. The AWS Payment Cryptography is a sub-CA of the manufacturer's CA that is the root of trust for HSM manufacturer and device certificates. The manufacturer's CA implements ANSI TR 34 and has attested compliance with PCI PIN Security Annex A and PCI P2PE Annex A. The manufacturer verifies that all HSM with device keys signed by the AWS Payment Cryptography CA are shipped to AWS' designated receiver.

As required by PCI PIN Security, the manufacturer supplies a list of serial numbers via a different communication channel than the HSM shipment. These serial numbers are checked at each step in the process of HSM installation into AWS data centers. Finally, AWS Payment Cryptography operators validate the list of installed HSM against the manufacturer's list before adding the serial number to list of HSM permitted to receive AWS Payment Cryptography keys.

HSMs are in secure storage or under dual control at all times, which includes:

- Shipment from the manufacturer to an AWS rack assembly facility.
- During rack assembly.
- Shipment from the rack assembly facility to a data center.
- Receipt and installation into a data center secure processing room. HSM racks enforce dual control with card access-controlled locks, alarmed door sensors, and cameras.
- During operations.
- During decommissioning and destruction.

A complete chain-of-custody, with individual accountability, is maintained and monitored for each HSM.

HSM initialization

An HSM is only initialized as part of the AWS Payment Cryptography fleet after its identity and integrity are validated by serial numbers, manufacturer installed device keys, and firmware checksum. After the authenticity and integrity of an HSM validated, it is configured, including enabling PCI Mode. Then AWS Payment Cryptography region main keys and profile main keys are established and the HSM is available to the service.

HSM service and repair

HSM have serviceable components that do not require violation of the device's cryptographic boundary. These components include cooling fans, power supplies, and batteries. If an HSM or another device within the HSM rack needs service, dual control is maintained during the entire period that the rack is open.

HSM decommissioning

Decommissioning occurs due to end-of-life or failure of an HSM. HSM are logically zero-ized before removal from their rack, if functional, then destroyed within secure processing rooms of AWS data centers. They are never returned to the manufacturer for repair, used for another purpose, or otherwise removed from a secure processing room before destruction.

HSM firmware update

HSM firmware updates are applied when required to maintain alignment with PCI PTS HSM and FIPS 140-2 (or FIPS 140-3) listed versions, if an update is security related, or it is determined that customers can benefit from features in a new version. AWS Payment Cryptography HSMs run off-the-shelf firmware, matching PCI PTS HSM-listed versions. New firmware versions are validated for integrity with the PCI or FIPS certified firmware versions then tested for functionality before rollout to all HSMs.

Operator access

Operators can have non-console access to HSM for troubleshooting in rare cases that information gathered from HSM during normal operations is insufficient to identify a problem or plan a change. The following steps are executed:

- Troubleshooting activities are developed and approved and the non-console session is scheduled.
- An HSM is removed from customer processing service.
- Main keys are deleted, under dual control.
- Operator is permitted non-console access to the HSM to perform approved troubleshooting activities, under dual control.
 - After termination of the non-console session, the initial provisioning process is performed on the HSM, returning the standard firmware and configuration, then synchronizing the main key, before returning the HSM to serving customers.

- Records of the session are recorded in change tracking.
- Information obtained from the session is used for planning future changes.

All non-console access records are reviewed for process compliance and potential changes to HSM monitoring, the non-console-access management process, or operator training.

Key management

All HSM in a region are synchronized to a Region Main Key. A Region Main Key protects at least one Profile Main Key. A Profile Main Key protects customer keys.

All main keys are generated by an HSM and distributed to by symmetric key distribution using asymmetric techniques, aligned with ANSI X9 TR 34 and PCI PIN Annex A.

Topics

- [Generation](#)
- [Region main key synchronization](#)
- [Region main key rotation](#)
- [Profile main key synchronization](#)
- [Profile main key rotation](#)
- [Protection](#)
- [Durability](#)
- [Communication security](#)
- [Management of customer keys](#)
- [Logging and monitoring](#)

Generation

AES 256 bit Main keys are generated on one of the HSM provisioned for the service HSM fleet, using the PCI PTS HSM random number generator.

Region main key synchronization

HSM region main keys are synchronized by the service across the regional fleet with mechanisms defined by ANSI X9 TR-34, which include:

- Mutual authentication using key distribution host (KDH) and key receiving device (KRD) keys and certificates to provide authentication and integrity of for public keys.
- Certificates are signed by a certificate authority (CA) that meets the requirements of PCI PIN Annex A2, except for asymmetric algorithms and key strengths appropriate for protecting AES 256 bit keys.
- Identification and key protection for the distributed symmetric keys consistent with ANSI X9 TR-34 and PCI PIN Annex A1, except for asymmetric algorithms and key strengths appropriate for protecting AES 256 bit keys.

Region main keys are established for HSMs that have been authenticated and provisioned for a region by:

- A main key is generated on an HSM in the region. That HSM is designated as the key distribution host.
- All provisioned HSMs in the region generate KRD authentication token, which contain the public key of the HSM and non-replayable authentication information.
- KRD tokens are added to the KDH allow list after the KDH validates the identity and permission of the HSM to receive keys.
- The KDH produces an authenticable main key token for each HSM. Tokens contain KDH authentication information and encrypted main key that is loadable only on an HSM that it has been created for.
- Each HSM is sent the main key token built for it. After validating the HSM's own authentication information and the KDH authentication information, the main key is decrypted by the KRD private key and loaded into the main key.

In the event that a single HSM must be re-synchronized with a region:

- It is re-validated and provisioned with firmware and configuration.
- If it is new to the region:
 - The HSM generates a KRD authentication token.
 - The KDH adds the token to its allow list.
 - The KDH generates a main key token for the HSM.
 - The HSM loads the main key.
 - The HSM is made available to the service.

This assures that:

- Only HSM validated for AWS Payment Cryptography processing within a region can receive that region's master key.
- Only a master key from an AWS Payment Cryptography HSM can be distributed to an HSM in the fleet.

Region main key rotation

Region main keys are rotated at the expiration of the crypto period, in the unlikely event of a suspected key compromise, or after changes to the service that are determined to impact the security of the key.

A new region main key is generated and distributed as with initial provisioning. Saved profile main keys must be translated to the new region main key.

Region main key rotation does not impact customer processing.

Profile main key synchronization

Profile main keys are protected by region main keys. This restricts a profile to a specific region.

Profile main keys are provisioned accordingly:

- A profile main key is generated on an HSM that has the region main key synchronized.
- The profile main key is stored and encrypted with the profile configuration and other context.
- The profile is used for customer cryptographic functions by any HSM in the region with the region main key.

Profile main key rotation

Profile main keys are rotated at the expiration of the crypto period, after suspected key compromise, or after changes to the service that are determined to impact the security of the key.

Rotation steps:

- A new profile main key is generated and distributed as a pending main key as with initial provisioning.

- A background process translates customer key material from the established profile main key to the pending main key.
- When all customer keys have been encrypted with the pending key, the pending key is promoted to the profile main key.
- A background process deletes customer key material protected by the expired key.

Profile main key rotation does not impact customer processing.

Protection

Keys depend only on the key hierarchy for protection. Protection of main keys is critical to prevent loss or compromise all customer keys.

Region main keys are restorable from backup only to HSM authenticated and provisioned for the service. These keys can only be stored as mutually authenticable, encrypted main key tokens from a specific KDH for a specific HSM.

Profile master keys are stored with profile configuration and context information encrypted by region.

Customer keys are stored in key blocks, protected by a profile master key.

All keys exist exclusively within an HSM or stored protected by another key of equal or stronger cryptographic strength.

Durability

Customer keys for transaction cryptography and business functions must be available even in extreme situations that would typically cause outages. AWS Payment Cryptography utilizes a multiple level redundancy model across availability zones and AWS regions. Customer's requiring higher availability and durability for payment cryptographic operations than what is provided by the service should implement multi-region architectures.

HSM authentication and main key tokens are saved and may be used to restore a main key or synchronize with a new main key, in the event that an HSM must be reset. The tokens are archived and used only under dual control when required.

Communication security

External

AWS Payment Cryptography API endpoints meet AWS security standards including TLS at or above 1.2 and Signature Version 4 for authentication and integrity of requests.

Incoming TLS connections are terminated on network load balancers and forwarded to API handlers over internal TLS connections.

Internal

Internal communications between service components and between service components and other AWS service are protected by TLS using strong cryptography.

HSM are on a private, non-virtual network that is only reachable from service components. All connections between HSM and service components are secured with mutual TLS (mTLS), at or above TLS 1.2. Internal certificates for TLS and mTLS are managed by Amazon Certificate Manager using an AWS Private Certificate Authority. Internal VPCs and the HSM network are monitored for unexcepted activities and configuration changes.

Management of customer keys

At AWS, customer trust is our top priority. You maintain full control of your keys that you upload or create in the service under your AWS account and responsibility for configuring access to keys.

AWS Payment Cryptography has full responsibility for the HSM physical compliance and key management for keys managed by the service. This requires ownership and management of HSM main keys and storage of protected customer keys within the AWS Payment Cryptography key database.

Customer key space separation

AWS Payment Cryptography enforces key policies for all key use, including limiting principals to the account owning the key, unless a key is explicitly shared with another account.

Backup and recovery

Keys and key information for a region is backed up to encrypted archives by AWS. Archives require dual control by AWS to restore.

Key blocks

All keys are stored in ANSI X9 TR-31 format key blocks.

Keys may be imported into the service from cryptograms or other key block formats supported by `ImportKey`. Similarly, keys may be exported, if they are exportable, to other key block formats or cryptograms supported by key export profiles.

Key use

Key use is restricted to the configured `KeyUsage` by the service. The service will fail any requests with inappropriate key usage, mode of use, or algorithm for the requested cryptographic operation.

Key exchange relationships

PCI PIN Security and PCI P2PE require that organizations that share keys that encrypt PINs, including the KEK used to share those keys, not share those keys with any other organizations. It is a best practice that symmetric keys are shared between only 2 parties, including within the same organization. This minimizes the impact of suspected key compromises that force replacing impacted keys.

Even business cases that require sharing keys between more than 2 parties, should keep the number of parties to the minimum number.

AWS Payment Cryptography provides key tags that can be used to track and enforce key usage within those requirements.

For example, KEK and BDK for different key injection facilities can be identified by setting a `"KIF"="POSSStation"` for all keys shared with that service provider. Another example would be to tag keys shared with payment networks with `"Network"="PayCard"`. Tagging enables you to create access controls and create audit reports to enforce and demonstrate your key management practices.

Key deletion

`DeleteKey` marks keys in the database for deletion after a customer-configurable period. After this period the key is irretrievably deleted. This is a safety mechanism to prevent the accidental or malicious deletion of a key. Keys marked for deletion are not available for any actions except `RestoreKey`.

Deleted keys remain in service backups for 7 days after deletion. They are not restorable during this period.

Keys belonging to closed AWS accounts are marked for deletion. If the account is reactivated before the deletion period is reached any keys marked for deletion are restored, but disabled. They must be re-enabled by you in order to use them for cryptographic operations.

Logging and monitoring

Internal service logs include:

- CloudTrail logs of AWS service calls made by the service
- CloudWatch logs of both events directly logged to CloudWatch logs or events from HSM
- Log files from HSM and service systems
- Log archives

All log sources monitor and filter for sensitive information, including about keys. Logs are systematically reviewed to ensure that they contain do not contain sensitive customer information.

Access to logs is restricted to individuals needed for completing job roles.

All logs are retained in alignment with AWS log retention policies.

Customer operations

AWS Payment Cryptography has full responsibility for the HSM physical compliance under PCI standards. The service also provides a secure key store and ensures that keys can only be used for the purposes permitted by PCI standards and specified by you during creation or import. You are responsible for configuring key attributes and access to leverage the security and compliance capabilities of the service.

Topics

- [Generating keys](#)
- [Importing keys](#)
- [Exporting keys](#)
- [Deleting keys](#)
- [Rotating keys](#)

Generating keys

When creating keys, you set the attributes that the service uses to enforce compliant use of the key:

- Algorithm and key length
- Usage
- Availability and expiration

Tags that are used for attribute-based access control (ABAC) are used to limit keys for use with specific partners or applications should also be set during creation. Be sure to include policies to limit roles permitted to delete or change tags.

You should ensure that the policies that determine the roles that may use and manage the key are set prior to the creation of the key.

Note

IAM policies on the CreateKey commands may be used to enforce and demonstrate dual control for key generation.

Importing keys

When importing keys, the attributes to enforce compliant use of the key are set by the service using the cryptographically bound information in the key block. The mechanism for setting fundamental key context is to use key blocks created with the source HSM and protected by a shared or asymmetric [KEK](#). This aligns with PCI PIN requirements and preserves usage, algorithm, and key strength from the source application.

Important key attributes, tags, and access control policies must be established on import in addition to the information in the key block.

Importing keys using cryptograms does not transfer key attributes from the source application. You must set the attributes appropriately by using this mechanism.

Often keys are exchanged using clear text components, transmitted by key custodians, then loaded with ceremony implementing dual control in a secure room. This is not directly supported by AWS

Payment Cryptography. The API will export a public key with a certificate that can be imported by your own HSM to export a key block that is importable by the service. The enables use of your own HSM for loading clear text components.

You should use Key check values (KCV) to verify that imported keys match source keys.

IAM policies on the ImportKey API may be used to enforce and demonstrate dual control for key import.

Exporting keys

Sharing keys with partners or on-premises applications may require exporting keys. Using key blocks for exports maintains fundamental key context with the encrypted key material.

Key tags can be used to limit the export of keys to KEK that share the same tag and value.

AWS Payment Cryptography does not provide or display clear text key components. This requires direct access by key custodians to PCI PTS HSM or ISO 13491 tested secure cryptographic devices (SCD) for display or printing. You can establish an asymmetric KEK or a symmetric KEK with your SCD to conduct the clear text key component creation ceremony under dual control.

Key check values (KCV) should be used to verify that imported by the destination HSM match source keys.

Deleting keys

You can use the delete key API to schedule keys for deletion after a period of time that you configure. Before that time keys are recoverable. Once keys are deleted they are permanently removed from the service.

IAM policies on the DeleteKey API may be used to enforce and demonstrate dual control for key deletion.

Rotating keys

The effect of key rotation can be implemented using key alias by creating or importing a new key, then modifying the key alias to refer to the new key. The old key would be deleted or disabled, depending on your management practices.

Quotas for AWS Payment Cryptography

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is region-specific. You can request increases for some quotas, and other quotas cannot be increased.

Name	Default	Adjustable	Description
Aliases	Each supported Region: 2,000	Yes	The maximum number of aliases you can have in this account in the current Region.
Combined rate of control plane requests	Each supported Region: 5 per second	Yes	The maximum number of control plane requests per second that you can make in this account in the current Region. This quota applies to all control plane operations combined.
Combined rate of data plane requests (asymmetric)	Each supported Region: 20 per second	Yes	The maximum number of requests per second for data plane operations with an asymmetric key that you can make in this account in the current Region. This quota applies to all data plane operations combined.
Combined rate of data plane requests (symmetric)	Each supported Region: 500 per second	Yes	The maximum number of requests per second for data plane operation

Name	Default	Adjustable	Description
			s with a symmetric key that you can make in this account in the current Region. This quota applies to all data plane operations combined.
Keys	Each supported Region: 2,000	Yes	The maximum number of keys you can have in this account in the current Region, excluding deleted keys.

Document history for the AWS Payment Cryptography User Guide

The following table describes the documentation releases for AWS Payment Cryptography.

Change	Description	Date
New region launch	Added endpoints for new region launch in Europe (Frankfurt), Europe (Ireland) , Asia Pacific (Singapore) and Asia Pacific (Tokyo)	July 31, 2024
CloudTrail for Dataplane and Dynamic Keys	Added information about utilizing CloudTrail for dataplane (cryptographic) operations including examples. Also added information about utilizing Dynamic Keys for certain functions to better support one-time or limited use keys that should not be imported into AWS Payment Cryptography	July 10, 2024
Updated Examples	Added new examples for card issuing	July 1, 2024
Feature release	Adding information on VPC endpoints(PrivateLink) and iCVV examples.	May 30, 2024
Feature release	Information added on new features around key import/export using RSA and	January 15, 2024

exporting DUKPT IPEK/IK
keys.

[Initial release](#)

Initial release of the AWS
Payment Cryptography User
Guide

June 8, 2023