AWS Key Management Service Cryptographic Details

August 2016

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

AWS Key Management Service (AWS KMS) provides cryptographic keys and operations scaled for the cloud. AWS KMS keys and functionality are used by other AWS cloud services, and you can use them to protect user data in your applications. This white paper provides details on the cryptographic operations that are executed within AWS when you use AWS KMS.

Introduction

AWS KMS provides a simple web services interface that can be used to generate and manage cryptographic keys and operate as a cryptographic service provider for protecting data. AWS KMS offers traditional key management services integrated with AWS services to provide a consistent view of customers' keys across AWS, with centralized management and auditing. This white paper supplies you with a detailed description of the cryptographic operations of AWS KMS to assist you in evaluating the features offered by the service.

AWS KMS provides a simple web interface in the AWS Management Console, Command Line Interface, and RESTful APIs to access an elastic, multi-tenant, hardened security appliance (HSA). You are able to establish your own HSA-based cryptographic contexts under your master keys. These keys are accessible only on the HSAs, and they can be used to perform HSA-resident cryptographic operations, including the issuance of application data keys (encrypted under your master key). You can create multiple master keys, each represented with an HSA-based customer master key (CMK) identified by its keyId. You can use the AWS KMS console to define access controls on who can manage and/or use master keys by creating a policy that is attached to the key. This allows you to define application-specific uses for your keys on a per-API basis.

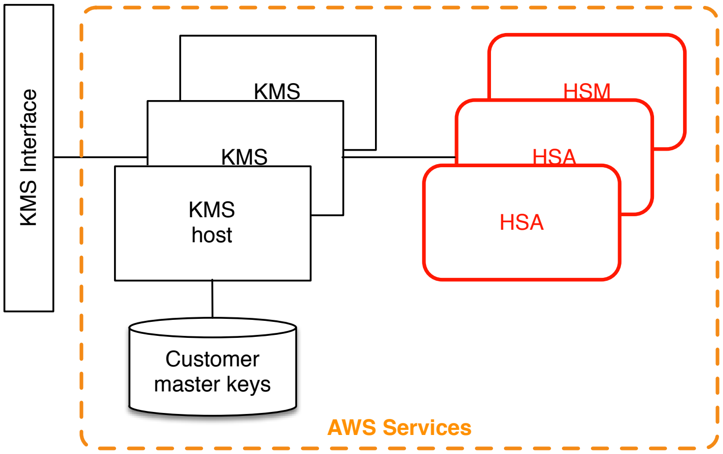

Figure 1: AWS KMS architecture

AWS KMS is a tiered service consisting of web-facing KMS hosts and a tier of HSAs. The grouping of these tiered hosts forms the AWS KMS stack. All requests to AWS KMS must be made over the Transport Layer Security protocol (TLS) and terminate on an AWS KMS host. AWS KMS hosts will only allow TLS with a ciphersuite that provides perfect forward secrecy. The AWS KMS hosts use protocols and procedures defined within this white paper to fulfill those requests through the HSAs. AWS KMS authenticates and authorizes customer requests using the same credential and policy mechanisms available for all other AWS APIs, including AWS Identity and Access Management (IAM).

Design Goals

AWS KMS is designed to meet the following requirements.

Durability: The durability of cryptographic keys is designed to equal that of the highest durability services in AWS. A single cryptographic key can encrypt large volumes of customer data accumulated over a long time period. However, data encrypted under a key becomes irretrievable if the key is lost.

Quorum-based access: No single Amazon employee can gain access to CMKs. There is no mechanism to export plaintext CMKs. Confidentiality of your cryptographic keys is crucial.

Access control: Use of keys is protected by access control policies defined and managed by you.

Low-latency and high throughput: AWS KMS will provide cryptographic operations at latency and throughput levels suitable for use by other services in AWS.

Regional independence: AWS provides regional independence for customer data. Key usage is isolated within an AWS Region.

Secure source of random numbers: Because strong cryptography depends on truly unpredictable random number generation, AWS provides a high-quality source of random numbers.

Audit: AWS records the use of cryptographic keys in AWS CloudTrail logs. Customers can use AWS CloudTrail logs to inspect use of their cryptographic keys, including use of keys by AWS services on the customer's behalf.

To achieve these goals, the AWS KMS system includes a set of KMS operators and service host operators (collectively, "operators") that administer "domains." A domain is a regionally defined set of AWS KMS servers, HSAs, and operators. Each entity has a hardware token that contains a private and public key pair used to authenticate its actions. The HSAs have an additional private and public key pair used to establish encryption keys to protect HSA-to-HSA communications.

This white paper illustrates how the AWS KMS protects your encryption keys and other data you want to encrypt. Throughout this document, we refer to either encryption keys or data you want to encrypt as "secrets" or "secret material."

Background

This section contains a description of the cryptographic primitives and where they are used. In addition, it introduces the basic elements of AWS KMS.

Cryptographic Primitives

AWS KMS uses configurable cryptographic algorithms so that the system can quickly migrate from one algorithm, or mode, to another. The initial default set of cryptographic algorithms has been selected from Federal Information Processing Standard (FIPS-approved algorithms) for their security properties and performance.

Entropy and Random Number Generation

AWS KMS key generation is performed on dedicated HSAs. These are physical devices without a virtualization layer, such as a hypervisor that would share the physical device among several logical tenants. The HSAs implement a hybrid random number generator. An initial cryptographically secure pseudo-random number generator (CSPRNG) is seeded with system entropy and periodically updated with additional entropy. Calls for cryptographic material use this hybrid random number generator.

Encryption

All symmetric key encrypt commands used within the HSA use the Advanced Encryption Standards (AES) [7], in Galois Counter Mode (GCM) [9] using 256-bit keys. The analogous calls to decrypt use the inverse function.

AES-GCM is an authenticated encryption scheme. In addition to encrypting plaintext to produce ciphertext, it computes an authentication tag over the ciphertext and any additional data over which authentication is required (additionally authenticated data, or AAD). The authentication tag helps ensure that the data is from the purported source and that the ciphertext, and AAD, have not been modified.

Frequently, AWS omits the inclusion of the AAD in our descriptions, especially when referring to the encryption of data keys. It is implied by surrounding text in these cases that the structure to be encrypted is partitioned between the plaintext to be encrypted and the cleartext AAD to be protected.

The ImportKeyMaterial operation allows customers to import a keying material into AWS KMS. This keying material is encrypted using RSAES-PKCS1-v1_5 or RSAES-OAEP [13]. The RSA key pairs are generated on an HSA. The imported key material is decrypted on an HSA, and encrypted under AES-GCM before being stored in our data storage layer.

Digital Signatures

There are two digital signature schemes utilized in AWS KMS: the elliptic curve digital signature algorithm (ECDSA) and RSA. All service host entities have an elliptic curve digital signature algorithm (ECDSA) key pair. They perform ECDSA as defined in Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)[14] and X9.62-2005: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)[3] using the secure hash algorithm defined in Federal Information Processing Standards Publications, FIPS PUB 180-4 [5], known as SHA384. The keys are generated on the curve secp384r1 (NIST-P384) [15].

AWS KMS operator entities use RSA-2048 key pairs using the RSA-PSS signature [13] using SHA256 [5].

Digital signatures are used to authenticate commands and communications between AWS KMS and operators. We denote a key pair as *(d, Q)*, the signing operation *Sig = Sign(d, msg)* as the sign operation and the verify operation *Verify(Q, msg, Sig),* which returns an indication of success or failure.

It will frequently be convenient to represent an entity by its public key *Q*. In these cases, we assume that identifying information, such as an identifier or a role, accompanies the public key.

Key Establishment

Two different key establishment methods are used in AWS KMS. The first is defined as C(1, 2, ECC CDH) in Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revision 2) [11]. This scheme has an initiator, with a static signing key, and the initiator generates and signs an ephemeral elliptic curve Diffie-Hellman (ECDH) key, intended for a second recipient with a static ECDH agreement key. This method uses one ephemeral key and two static keys using ECDH, and that is the derivation of the label C(1, 2, ECC CDH). This method is sometimes called one-pass ECDH.

The second key establishment method is C(2, 2, ECC, CDH) [11]. In this scheme, both parties have a static signing key, and they generate, sign, and exchange an ephemeral ECDH key. This method uses two static keys and two ephemeral keys using ECDH, and that is the derivation of the label C(2, 2, ECC CDH). This method is sometimes called ECDH ephemeral or ECDHE. All ECDH keys are generated on the curve secp384r1 (NIST-P384) [15].

Envelope Encryption

A basic construction used within many cryptographic systems is envelope encryption. Envelope encryption utilizes two or more cryptographic keys to secure a message. Typically, one of the keys is derived from a longer-term static key *k,* and one of the keys is a per-message key, *msgKey*, generated to encrypt the message. The envelope is formed by encrypting the message, *ciphertext* = *Encrypt(msgKey, message)*, encrypting the message key with the long-term static key, *encKey* = *Encrypt(k, msgKey)*, and packaging the two values *(encKey, ciphertext)* into a single structure, or envelope encrypted message.

The recipient, with access to *k*, can open the enveloped message by first decrypting the encrypted key and then decrypting the message.

AWS KMS provides you with the ability to manage these longer static keys and automate the process of envelope encryption of customer data.

AWS KMS uses envelope encryption internally to secure confidential material between service endpoints.

The AWS Encryption SDK for Java [16] provides client-side envelope encryption libraries you can use to protect your data and the encryption keys used to encrypt that data.

Basic Concepts

This section introduces some basic AWS KMS concepts that are elaborated on throughout this white paper.

Customer master key (CMK): A logical key that represents the top of a customer's key hierarchy. A CMK is given an Amazon Resource Name (ARN) that includes a unique key identifier, or *keyID*.

Alias: A user-friendly name, or *alias*, can be associated with a CMK. The alias can be used interchangeably with *keyID* in many of the AWS KMS APIs.

Permissions: A policy attached to a CMK that defines permissions on the key. The default policy enables any principals you define, as well as enabling the root user in the account to add IAM policies that reference the key.

Grants: Grants are intended to allow delegated use of CMKs when the duration of usage is not known at the outset. One use of grants is to define scoped-down permissions for an AWS service to use your key to do asynchronous work on your behalf on encrypted data in the absence of a direct-signed API call from you.

Data keys: Cryptographic keys generated on an HSA under a CMK. AWS KMS allows authorized entities to obtain data keys protected by a CMK. They can be returned both as plaintext (unencrypted) data keys and as encrypted data keys.

Ciphertexts: We refer to the encrypted output of AWS KMS as *customer ciphertext* or just *ciphertext* when there is no confusion. Ciphertext contains encrypted data with additional information that identifies the CMK to use in the decryption process.

Encryption context: A key-value pair map of additional information associated with AWS KMS-protected information. AWS KMS uses authenticated encryption to protect data keys. The encryption context is incorporated into the additional authenticated data (AAD) of the authenticated encryption in AWS KMS-encrypted ciphertexts. This context information is optional and not returned when requesting a key (or an encryption operation) but, if used, this context value will be required to successfully complete a decryption operation. An intended use of the encryption context is to provide additional authenticated information that can be used to enforce policies, and be included in the AWS CloudTrail logs. For example, a key-value pair of {"key name":"satellite uplink key"} could be used to name the data key. Subsequently, whenever the key is used the AWS CloudTrail entry will be made that includes "key name": "satellite uplink key." This additional information can provide useful context to understand why a given master key was used.

Customer's Key Hierarchy

The customer's key hierarchy starts with a top-level logical key, a CMK. A CMK represents a container for top-level key material and is uniquely defined within the AWS service namespace with an ARN. The ARN will include a uniquely generated key identifier, a CMK keyID. A CMK is created based on a user-initiated request through AWS KMS. Upon reception, AWS KMS will request the creation of an initial HSA backing key (HBK) to be placed into the CMK container. We denote all such HSA-resident-only keys in red. The HBK will be generated on an HSA in the domain and is designed to never be exported from the HSA in plaintext. Instead, the HBK is exported encrypted under HSA-managed domain keys. We refer to these exported HBKs as exported key tokens (EKT).

The EKT is exported to highly durable, low-latency storage. The customer will receive an ARN to the logical CMK. This represents the top of a key hierarchy, or cryptographic context, for the customer. You can create multiple CMKs within your account and set policies on your CMKs like any other AWS-named resource.

Within the hierarchy of a specific CMK, the HBK can be thought of as a version of the CMK. When a customer wants to rotate the CMK through AWS KMS a new HBK is created and associated with the CMK as the active HBK for the CMK. The older HBKs are preserved and can be used to decrypt and verify previously protected information, but only the active cryptographic key can be used to protect new information.

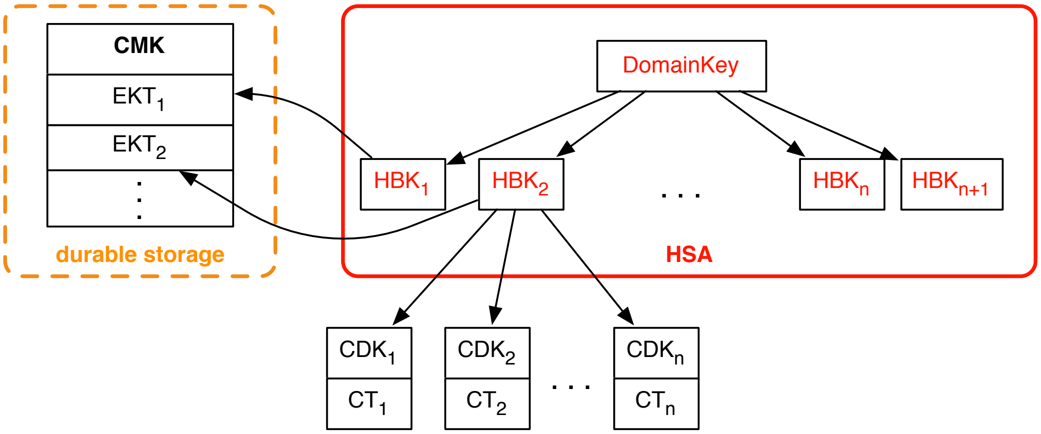

Figure 2: CMK hierarchy

You can make requests through AWS KMS to use your CMKs to directly protect information or request additional HSA-generated keys protected under your CMK. These keys are called customer data keys, or CDKs. CDKs can be returned encrypted, in plaintext, or both. All HSA-encrypted objects under a CMK (either customer-supplied data or HSA-generated) can be decrypted only on an HSA via a call through AWS KMS.

The returned ciphertext, or the decrypted payload, is never stored within AWS KMS. The only copy of this information is returned to you over your TLS connection to AWS KMS. This also applies to calls made by AWS services on your behalf.

HSA-provided schemes currently support direct encryption and symmetric-key authentication schemes. The architecture allows for future expansion to offer additional cryptographic services.

We summarize the key hierarchy and the specific key properties in the following table.

Key	Description	Lifecycle

Key	Description	Lifecycle
Domain Key	A 256-bit AES-GCM key only in memory of HSA, used to wrap HSA Backing Keys.	Rotated daily[1]
HSA Backing Key	A 256-bit symmetric key only in memory of HSA used to protect customer data keys. Stored encrypted under Domain Keys	Rotated yearly[2] (optional config.)
Customer Data Key	User-defined key exported from HSA in plaintext and ciphertext, encrypted under an HSA Backing Key, and returned to authorized users over TLS channel.	Rotation and use controlled by application

Use Cases

This white paper presents two use cases. The first demonstrates how AWS KMS performs server-side encryption with CMKs on an Amazon Elastic Block Store (Amazon EBS) volume. The second is a client-side application that demonstrates how customers can use envelope encryption to protect content with AWS KMS.

Amazon EBS Volume Encryption

Amazon EBS offers a volume encryption. Each volume is encrypted using AES-256-XTS [10]. This requires two 256-bit volume keys, which you can think of as one 512-bit volume key. The volume key is encrypted under a CMK in your account. For Amazon EBS to encrypt a volume for you, it must have access to generate a volume key (VK) under a CMK in the account. You do this by providing a grant for Amazon EBS to the CMK to create data keys, and to encrypt and decrypt these volume keys. Now, Amazon EBS will use AWS KMS with a CMK to generate AWS KMS–encrypted volume keys.

[1] AWS KMS may from time to time relax domain key rotation to at most weekly to account for domain administration and configuration tasks.

[2] Default service master keys created and managed by AWS KMS on your behalf are automatically rotated every 3 years.

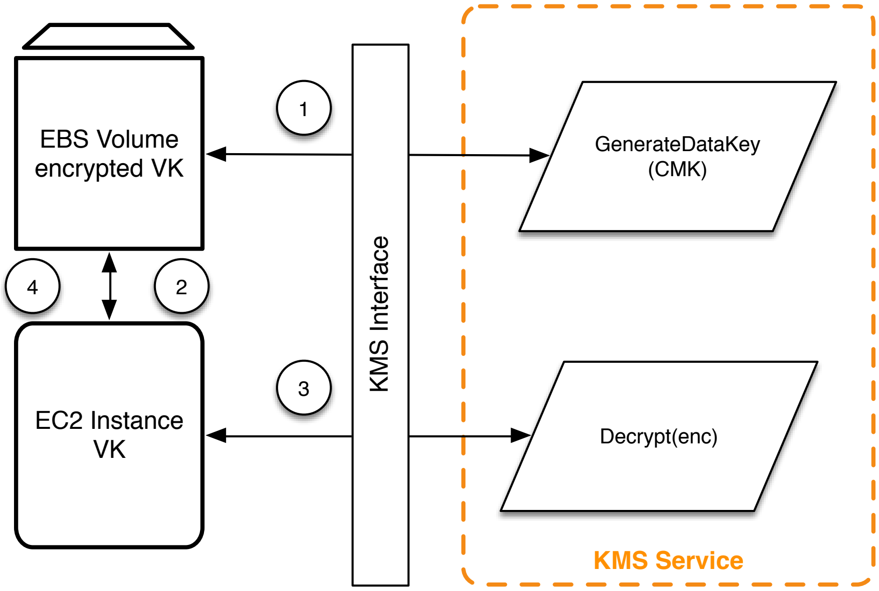

Figure 3: Amazon EBS volume encryption with AWS KMS keys

The basic steps to encrypt data being written to an EBS volume are:

1. Amazon EBS obtains an encrypted volume key under a CMK through AWS KMS over a TLS session, and stores the encrypted key with the volume metadata.
2. When the EBS volume is mounted, the encrypted volume key is retrieved.
3. A call to AWS KMS over TLS is made to decrypt the encrypted volume key. AWS KMS will identify the CMK and make an internal request to an HSA in the fleet to decrypt the encrypted volume key, and will return the volume key back to the Amazon Elastic Compute Cloud (Amazon EC2) host that contains your instance over the TLS session.
4. The volume key is used to encrypt and decrypt all data going to and from the attached EBS volume. Amazon EBS retains the encrypted volume key for later use in case the volume key in memory is no longer available.

Envelope Encryption

The AWS Encryption SDK [16] includes an API for performing envelope encryption using a CMK from AWS KMS. For complete recommendations and usage details see the related documentation [16]. Client applications can use the AWS Encryption SDK to perform envelope encryption using AWS KMS.

```
// Instantiate the SDK
final AwsCrypto crypto = new AwsCrypto();
// Set up the KmsMasterKeyProvider backed by the default credentials
final KmsMasterKeyProvider prov = new KmsMasterKeyProvider(keyId);
// Do the encryption
final byte[] ciphertext = crypto.encryptData(prov, message);
```

The client application can execute the following steps:

1. A request that is made under a CMK for a new data key. An encrypted data key and a plaintext version of the data key are returned.
2. Within the AWS Encryption SDK the plaintext data key is used to encrypt the message, and then the plaintext data key is deleted from memory.
3. The encrypted data key and encrypted message are combined into a single ciphertext byte array.

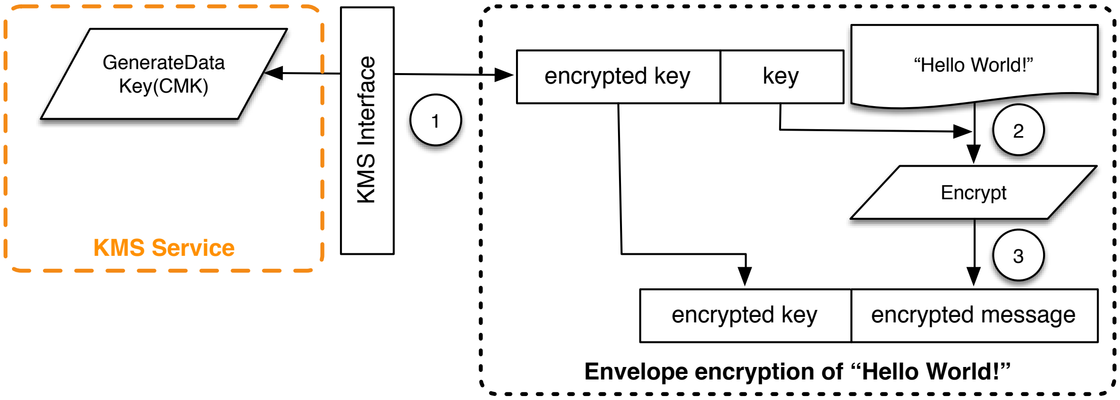

Figure 4: AWS Encryption SDK envelope encryption

The envelope-encrypted message can be decrypted using the decrypt functionality to obtain the originally encrypted message.

```
final AwsCrypto crypto = new AwsCrypto();
final KmsMasterKeyProvider prov = new KmsMasterKeyProvider(keyId);
// Decrypt the data
final CryptoResult<byte[], KmsMasterKey> res =
crypto.decryptData(prov, ciphertext);
// We need to check the master key to ensure that the
// assumed key was used
if (!res.getMasterKeyIds().get(0).equals(keyId)) {
     throw new IllegalStateException("Wrong key id!");
}
byte[] plaintext = res.getResult();
```

1. The AWS Encryption SDK will parse the envelope-encrypted message to obtain the encrypted data key and make a request to AWS KMS to decrypt the data key.
2. The AWS Encryption SDK will receive the plaintext data key from AWS KMS.
3. The data key is then used to decrypt the message, returning the initial plaintext.

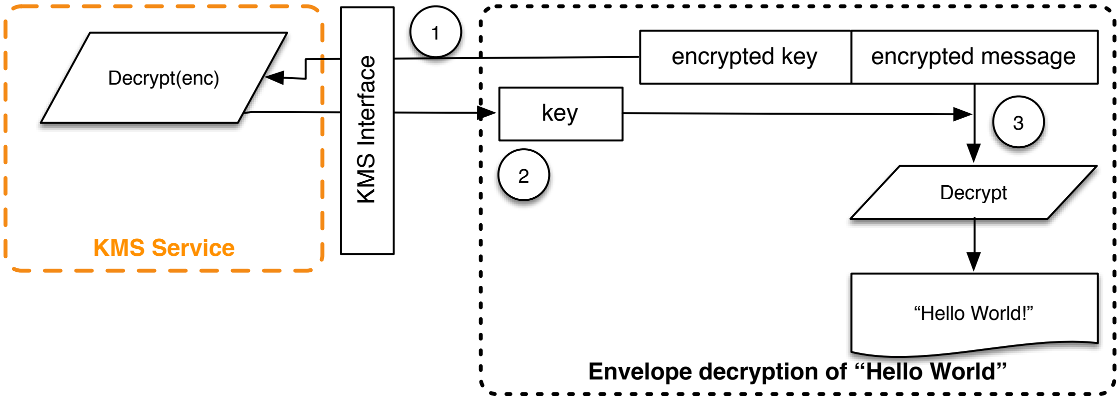

Figure 5: AWS Encryption SDK envelope decryption

Customer Master Keys

A CMK refers to a logical key that may refer to one or more HBKs. It is generated as a result of a call to the CreateKey API.

The following is the CreateKey request syntax.

```
{
  "Description": "string",
  "KeyUsage": "string",
  “Origin”: “string”;
  "Policy": "string"
}
```

The request accepts the following data in JSON format.

Optional Description: Description of the key. We recommend that you choose a description that helps you decide whether the key is appropriate for a task.

Optional KeyUsage: Specifies the intended use of the key. Currently this defaults to “ENCRYPT/DECRYPT”, and only symmetric encryption and decryption are supported.

Optional Origin: The source of the CMK's key material. The default is “AWS_KMS”. In addition to the default value “AMS_KMS”, the value “EXTERNAL” may be used to create a CMK without an HBK. The use of EXTERNAL is covered in the following section on Imported Master Keys.

Optional Policy: Policy to attach to the key. If the policy is omitted, the key will be created with the default policy (below) that enables IAM users with AWS KMS permissions, as well as the root account to manage it.

For details on the policy, see http://docs.aws.amazon.com/kms/latest/developerguide/key-policies.html.

The call will return a response containing an ARN with the key identifier.

```
arn:aws:kms:<region>:<owningAWSAccountId>:key/<keyId>
```

If the Origin is AWS_KMS, after the ARN is created, a request to an HSA is made over an authenticated session to provision an HBK, a 256-bit key that is associated with this CMK keyID. It can be generated only on an HSA and is designed to never be exported outside of the HSA boundary in clear text. An HBK is generated on the HSA and exported encrypted under the current domain key DK_0. We refer to these encrypted HBKs as EKTs. Although the HSAs can be configured to use a variety of key wrapping methods, the current implementation uses the authenticated encryption scheme known as AES-256 in Galois Counter Mode (GCM) [9]. As part of the authenticated encryption mode, we can protect some cleartext exported key token metadata.

We represent this stylistically as *EKT = Encrypt(*DK_0*, HBK)*.

There are two fundamental forms of protection provided to your CMKs and the subsequent HBKs, authorization policies set on your CMKs, and the cryptographic protections on your associated HBKs. The remaining sections describe the cryptographic protections and the security of the management functions utilized in AWS KMS.

In addition to the ARN, a customer friendly name can be associated with the CMK by creating an alias for the key. Once an alias has been associated with a CMK, the alias can be used in place of the ARN.

There are multiple levels of authorizations around the use of CMKs. AWS KMS enables separate authorization policies between the encrypted content and the CMK. For instance, an AWS KMS envelope-encrypted Amazon Simple Storage Service (Amazon S3) object will inherit the policy on the Amazon S3 bucket, while access to the necessary encryption key will be determined by the access policy on the CMK.

For the latest information about authentication and authorization policies for AWS KMS, see http://docs.aws.amazon.com/kms/latest/developerguide/control-access.html.

Imported Master Keys

AWS KMS provides a mechanism for importing the cryptographic material used for an HBK. As described in the section on Customer Master Keys earlier in this white paper, when the CreateKey command is used with Origin set to EXTERNAL, a logical CMK is created that contains no underlying HBK. The cryptographic material must be imported using the ImportKeyMaterial API. This feature allows customers to control the key creation and durability of the cryptographic material. It is recommended that users of this feature take significant caution in the handling and durability of these keys in their environment. For complete details and recommendations for importing master keys, see https://docs.aws.amazon.com/kms/latest/developerguide/importing-keys.html.

GetParametersForImport

Prior to importing the key material for an imported master key, the customer must obtain the necessary parameters to import the key.

The following is the GetParametersForImport request syntax.

```
{
  "KeyId": "string",
  "WrappingAlgorithm": "string",
  “WrappingKeySpec” : “string”
}
```

KeyId: A unique key identifier for a CMK. This value can be a globally unique identifier, an ARN, or an alias.

WrappingAlgorithm: The algorithm you will use when you encrypt your key material. The valid values are “RSAES_OAEP_SHA256”, “RSAES_OAEP_SHA1”, or “RSAES_PKCS1_V1_5”. AWS KMS recommends that you use RSAES_OAEP_SHA256. You may have to use another key-wrapping algorithm, depending on what your key management infrastructure supports.

WrappingKeySpec: The type of wrapping key (public key) to return in the response. Only RSA 2048-bit public keys are supported. The only valid value is "RSA_2048."

This call results in a request from the AWS KMS host to an HSA to generate a new RSA 2048-bit key pair to use for importing an HBK for the specified CMK keyId. The private key is protected and accessible only by an HSA member of the domain.

A successful call results in the following return values.

```
{
  "ImportToken": blob,
  "KeyId": "string",
  "PublicKey": blob,
  "ValidTo": number
}
```

ImportToken: A token that contains metadata to ensure that your key material is imported correctly. Store this value and send it in a subsequent ImportKeyMaterial request.

KeyId: The CMK to use when you subsequently import the key material. This is the same CMK specified in the request.

PublicKey: The public key to use to encrypt your key material. The public key is encoded as specified in section A.1.1 of PKCS#1 [13], an ASN.1 DER encoding of the RSAPublicKey. It is the ASN.1 encoding of two integers as an ASN.1 sequence.

ValidTo: The time at which the import token and public key expire. These items are valid for 24 hours. If you do not use them for a subsequent ImportKeyMaterial request within 24 hours, you must retrieve new ones. The import token and public key from the same response must be used together.

ImportKeyMaterial

The ImportKeyMaterial imports the necessary cryptographic material for the HBK. The cryptographic material must be a 256-bit symmetric key. It must be encrypted using the algorithm specified in WrappingAlgorithm under the returned public key from a recent GetParametersForImport request.

ImportKeyMaterial takes the following arguments.

```
{
  "EncryptedKey": blob,
  "ExpirationModel": "string",
  "ImportToken": blob,
  "KeyId": "string",
  "ValidTo": number
}
```

EncryptedKey: The encrypted key material. Encrypt the key material with the algorithm you specified in a previous GetParametersForImport request and the public key you received in the response to that request.

ExpirationModel: Specifies whether the key material expires. When this value is KEY_MATERIAL_EXPIRES, the ValidTo parameter must contain an expiration date. When this value is KEY_MATERIAL_DOES_NOT_EXPIRE, do not include the ValidTo parameter. The valid values are "KEY_MATERIAL_EXPIRES" and "KEY_MATERIAL_DOES_NOT_EXPIRE".

ImportToken: The import token you received in a previous GetParametersForImport response. Use the import token from the same response that contained the public key that you used to encrypt the key material.

KeyId: The CMK to import key material into. The CMK's Origin must be EXTERNAL.

Optional ValidTo: The time at which the imported key material expires. When the key material expires, AWS KMS deletes the key material and the CMK becomes unusable. You must omit this parameter when ExpirationModel is set to KEY_MATERIAL_DOES_NOT_EXPIRE, otherwise it is required.

On success, the CMK will be available for use within AWS KMS until the specified validity date. Once an imported CMK expires, the EKT is deleted from our storage layer. It can also be removed directly using the DeleteImportedKeyMaterial.

Enable and Disable Key

The ability to enable or disable a CMK is separate from the key lifecycle. This does not modify the actual state of the key, but instead suspends the ability to use all HBKs tied to a CMK. These are simple commands that take just the CMK keyID.

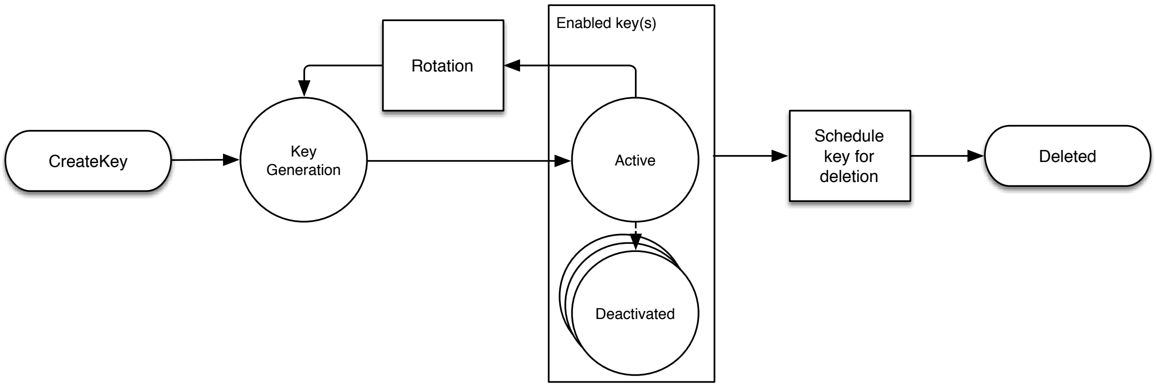

Figure 6: AWS KMS CMK lifecycle[3]

Key Deletion

Customers can delete a CMK and all associated HBKs. This is an inherently destructive operation and customers should exercise caution when deleting keys from KMS. AWS KMS enforces a minimal wait time of seven days when deleting CMKs. During the waiting period the key is placed in a disabled state with a key state indicating Pending Deletion. All calls to use the key for cryptographic

[3] The lifecycle for an EXTERNAL CMK differs. It can be in the state of pending import, and key rotation is not currently available. Further, the EKT can be removed without requiring a waiting period by calling DeleteImportedKeyMaterial.

operations will fail. CMKs can be deleted using the ScheduleKeyDeletion API. It takes the following arguments.

```
{
  "KeyId": “string”,
  "PendingWindowInDays": number
}
```

KeyId: The unique identifier for the CMK to delete. To specify this value, use the unique key ID or the ARN of the CMK.

Optional PendingWindowInDays: The waiting period, specified in number of days. After the waiting period ends, AWS KMS deletes the CMK and all associated HBKs. This value is optional. If you include a value, it must be between 7 and 30, inclusive. If you do not include a value, it defaults to 30.

Rotate Customer Master Key

Customers can induce a rotation of their CMK. The current system allows customers to opt in to a yearly rotation schedule to their CMK. When a CMK is rotated, a new HBK is created and marked as the active key for all new requests to protect information. The current active key is moved to the deactivated state and remains available for use to decrypt any existing ciphertext values that have been encrypted using this version of the HBK. AWS KMS does not store any ciphertext values encrypted under a CMK, as a direct consequence these ciphertext values require the deactivated HBK to decrypt. These older ciphertexts can be moved to the new HBK by calling the ReEncrypt API.

You can set up key rotation using a simple API call or by using the AWS Management Console.

Customer Data Operations

After a customer has established a CMK, it can be used to perform cryptographic operations. Whenever an element is encrypted under a CMK the resulting object is a customer ciphertext. The ciphertext will contain two sections: an unencrypted header (or cleartext) portion, protected by the authenticated encryption scheme as the additional authenticated data, and an encrypted portion. The cleartext portion will include the HSA backing key identifier (HBKID). These two immutable fields of the ciphertext value help ensure that AWS KMS will be able to decrypt the object in the future.

Application-Specific Data Keys

A request can be made to obtain application-specific data keys. A request can be made for a specific type of data key or a random key of arbitrary length through the GenerateDataKey API. We provide a simplified view of this API here and in other examples. You can find a detailed description of the full API here http://docs.aws.amazon.com/kms/latest/APIReference/Welcome.html**.**

The following is the GenerateDataKey request syntax.

```
{
   "EncryptionContext": {"string" : "string"},
   "GrantTokens": ["string"],
   "KeyId": "string",
   "KeySpec": "string",
   "NumberOfBytes": "number"
}
```

The request accepts the following data in JSON format.

Optional EncryptionContext: Name:value pair that contains additional data to authenticate during the encryption and decryption processes that use the key.

Optional GrantTokens: A list of grant tokens that represent grants that can be used to provide permissions to generate or use a key. For more information on grants and grant tokens, see http://docs.aws.amazon.com/kms/latest/developerguide/control-access.html.

Optional KeySpec: A value that identifies the encryption algorithm and key size. Currently this can be AES_128 or AES_256.

Optional NumberOfBytes: An integer that contains the number of bytes to generate.

AWS KMS, after authenticating the command, will acquire the current active EKT pertaining to the CMK. It will pass the EKT along with the customer-provided request and any encryption context to an HSA over a protected session between the AWS KMS host and an HSA in the domain.

The HSA will do the following:

1. Generate the requested secret material and hold it in volatile memory.
2. Decrypt the *EKT* matching the keyID of the CMK defined in the request to obtain the active $HBK = Decrypt(DK_i , EKT)$.
3. Determine a 256-bit encryption key K from HBK.
4. Encrypt the plaintext $ciphertext = Encrypt(K, context, secret)$.

The ciphertext value is returned to the customer, and is not retained anywhere in the AWS infrastructure. Without possession of the *ciphertext* and the encryption context, and the authorization to use the *CMK,* the underlying secret cannot be returned.

The GenerateDataKey returns the secret value and the ciphertext to the customer over the secure channel between the AWS KMS host and the HSA host, and then over the TLS session to the customer from AWS KMS.

The following is the response syntax.

```
{
   "CiphertextBlob": "blob",
   "KeyId": "string",
   "Plaintext": "blob"
}
```

The management of data keys is left to the application developer. They can be rotated at any frequency. Further, the data key itself can be re-encrypted to a different CMK or a rotated CMK using the ReEncrypt API. Full details can be found here: http://docs.aws.amazon.com/kms/latest/APIReference/Welcome.html.

Encrypt

A basic function of AWS KMS is to encrypt an object under a CMK. By design, AWS KMS provides low latency cryptographic operations on dedicated HSAs. A result of this is a limit of 4 KB on the amount of plaintext that can be encrypted in a direct call to the encrypt function. Envelope encryption is used to encrypt larger messages. AWS KMS, after authenticating the command, will acquire the current active EKT pertaining to the CMK. It will pass the EKT along with the customer-provided plaintext and encryption context to any available HSA in the region over an authenticated session between the AWS KMS host and an HSA in the domain.

The HSA will execute the following:

1. Decrypt the *EKT* to obtain the $HBK = Decrypt(DK_i , EKT)$.
2. Determine a 256-bit encryption key K from HBK.
3. Encrypt the plaintext $ciphertext = Encrypt(K, context, plaintext)$.

The ciphertext value is returned to the customer, and is not retained anywhere in the AWS infrastructure. Without possession of the *ciphertext* and the encryption context, and the authorization to use the CMK, the underlying plaintext cannot be returned.

Decrypt

A call to AWS KMS to decrypt a ciphertext value accepts an encrypted value ciphertext and an encryption context. AWS KMS will authenticate the call using AWS signature version 4 signed requests, and extract the HBKID for the wrapping key from the ciphertext. The HBKID is used to obtain the *EKT* required to decrypt the ciphertext, the keyId, and the policy for the keyId. The request is authorized based on the key policy, grants that may be present, and any associated IAM policies that reference the keyId. The Decrypt function is analogous to the encryption function.

The following is the Decrypt request syntax.

```
{
   "CiphertextBlob": "blob",
   "EncryptionContext": { "string" : "string" }
   "GrantTokens": ["string"]
}
```

The following are the request parameters.

CiphertextBlob: Ciphertext including metadata.

Optional EncryptionContext: The encryption context. If this was specified in the Encrypt function, it must be specified here or the decryption operation will fail. For more information, see https://docs.aws.amazon.com/kms/latest/developerguide/encrypt-context.html.

Optional GrantTokens: A list of grant tokens that represent grants that can be used to provide permissions to perform decryption.

The *ciphertext* and the *EKT* are sent, along with the encryption context, over an authenticated session to an HSA for decryption.

The HSA will execute the following:

1. Decrypt the *EKT* to obtain the *HBK* = *Decrypt*(DK_i , *EKT*).
2. Determine a 256-bit encryption key *K* from *HBK*.
3. Decrypt the *ciphertext* to obtain *plaintext* = *Decrypt(K, context, ciphertext)*.

The resulting keyId and plaintext is returned to the AWS KMS host over the secure session, and then back to the calling customer application over a TLS connection.

The following is the response syntax.

```
{
    "KeyId": "string",
    "Plaintext": blob
}
```

If the calling application wants to ensure the authenticity of the plaintext, it must verify the keyId returned is the one expected.

Re-Encrypting an Encrypted Object

An existing customer ciphertext encrypted under one CMK can be moved to another CMK through a re-encrypt command. Re-encrypt encrypts data on the server side with a new CMK without exposing the plaintext of the key on the client side. The data is first decrypted and then encrypted.

The following is the request syntax.

```
{
   "CiphertextBlob": "blob",
   "DestinationEncryptionContext": { "string" : "string" },
   "DestinationKeyId": "string",
   "GrantTokens": ["string"],
   "SourceEncryptionContext": { "string" : "string"}
}
```

The request accepts the following data in JSON format.

CiphertextBlob: Ciphertext of the data to re-encrypt.

Optional DestinationEncryptionContext: Encryption context to be used when the data is re-encrypted.

DestinationKeyId: Key identifier of the key used to re-encrypt the data.

Optional GrantTokens: A list of grant tokens that represent grants that can be used to provide permissions to perform decryption.

Optional SourceEncryptionContext: Encryption context used to encrypt and decrypt the data specified in the CiphertextBlob parameter.

The process combines the decrypt and encrypt of the previous descriptions, where the customer ciphertext is decrypted under the initial HBK referenced by the customer ciphertext to the current HBK under the second CMK. When the CMKs used in this command are the same, this command moves the customer ciphertext from an old version of an HBK to the latest version of an HBK.

Internal Communication Security

Commands between the service hosts/KMS operators, and the HSAs are secured through two mechanisms depicted in Figure-7: a quorum-signed request method and an authenticated session using an HSA-service host protocol.

The quorum-signed commands are designed so that no single operator can modify the critical security protections provided by the HSAs. The commands executed over the authenticated sessions help ensure that only authorized operators can perform operations involving cryptographic keys, and all customer-bound secret information is secured across the AWS infrastructure.

HSA Security Boundary

The inner security boundary of AWS KMS is the HSA. The HSA has a limited web-based API and no other active physical interfaces in its operational states. An operational HSA is provisioned during initialization with the necessary cryptographic keys. Sensitive cryptographic materials of the HSA are only stored in volatile memory, and erased when the HSA moves out of the operational state, including intended or unintended shutdowns or resets.

The HSA APIs are authenticated either by individual commands or over a mutually authenticated confidential session established by a service host.

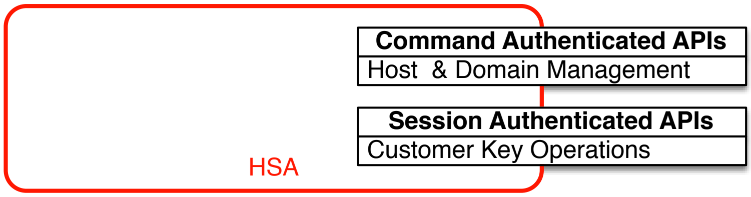

Figure 7: HSA APIs

Quorum-Signed Commands

Quorum-signed commands are issued by operators to HSAs. This section describes how quorum-based commands are created, signed, and authenticated. These rules are fairly simple, like Command *Foo* requires two members from Role *Bar* to authenticate. There are three steps in the creation and verification of a quorum-based command. The first step is the initial command creation, the second is the submission to additional operators to sign, and the third is the verification and execution.

For the purpose of introducing the concepts, we will assume we have an authentic set of operator's public keys and roles $\{QOS_s\}$, and a set of quorum-rules $QR = \{ Command_i , \{ Rule_{\{i, t\}}\}$ where each *Rule* is a set of roles and minimum number N $\{Role_t ,N_t \}$. For a command to satisfy the quorum rule, the command dataset must be signed by a set of operators listed in $\{QOS_s\}$ such that they meet one of the rules listed for that command. As mentioned earlier in this white paper, the set of quorum rules and operators are stored in the domain state and the exported domain token.

In practice, an initial signer signs the command $Sig_1 = Sign(d_{Op1}, Command)$. A second operator also signs the command

$Sig_2 = Sign(d_{Op2}, Command)$. The doubly signed message is sent to an HSA for execution. The HSA performs the following:

1. For each signature, it extracts the signer's public key from the domain state, and verifies the signature on the command.
2. It verifies that the set of signers satisfies a rule for the command.

Authenticated Sessions

This protocol performs a mutually authenticated ECDHE key agreement between the HSA and the service host. The exchange is initiated by the service host and completed by the HSA. The HSA also returns an exported key token that contains the negotiated session key. The exported key token contains a validity period, after which the service host must renegotiate a session key.

The customer key operations are executed between the externally facing AWS KMS hosts and the HSAs. These commands pertain to the creation and use of cryptographic keys and secure random number generation. The commands execute over a session-authenticated channel between the service hosts and the HSAs. In addition to the need for authenticity, these sessions require confidentiality. Commands executing over these sessions include the returning of cleartext data keys and decrypted messages intended for the customer. To help ensure that these sessions cannot be subverted through man-in-the-middle attacks, the sessions must also be authenticated.

A service host is a member of the domain, and has an identity signing key pair *($dHOS_i$, $QHOS_i$)* and an authentic copy of the HSAs' identity keys. It uses its set of identity keys to securely negotiate a session key that can be used between the service host and any HSA in the domain. The exported key tokens have a validity period associated with them, after which a new key must be negotiated.

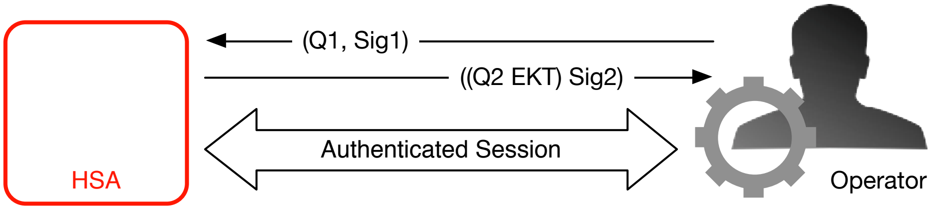

Figure 8: HSA-service host operator authenticated sessions

The process begins with the service host recognition that it requires a session key to send and receive sensitive communication flows between itself and an HSA member of the domain.

1. A service host generates an ECDH ephemeral key-pair *(d_1, Q_1)*, and signs it with its identity key $Sig_1 = Sign(dOS,Q_1)$.

2. The HSA verifies the signature on the received public key using its current domain token, creates an ECDH ephemeral key-pair (d_2, Q_2), completes the ECDH-key-exchange according to Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised) [11] to form a session key *SK,* and then wraps it as an exported key token *EKT* and signs a return value with its identity key pair $Sig_2 = Sign(dHSK, (Q_2, EKT))$.

3. The service host verifies the signature on the received key using its current domain token, and completes the ECDH key exchange according to Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised) [11] to form a session key *SK*.

During the validity period in the *EKT* the service host can use the negotiated session key *SK* to send envelope-encrypted commands to the HSA. Every service-host-initiated command over this authenticated session includes the *EKT*. The HSA will respond using the same negotiated session key SK.

Domains and the Domain State

We refer to a cooperative collection of trusted internal AWS KMS entities within an AWS Region as a domain. A domain includes a set of trusted entities, a set of rules, and a set of secret keys, called domain keys. The domain keys are shared among HSAs that are members of the domain. A domain state consists of the following fields.

Field	Description
Name	A domain name to identify this domain.
Members	A list of HSAs that are members of the domain, including their public signing key and public agreement keys.
Operators	A list of entities, public signing keys, and a role (KMS operator or service host) that represents the operators of this service.
Rules	A list of quorum rules for each command that must be satisfied to execute a command on the HSA.
Domain Keys	A list of domain keys (symmetric keys) currently in use within the domain.

Domain Keys

All the HSAs in a domain share a set of domain keys, *{DK_r}*. These keys are shared through a domain state export routine. The exported domain state can be imported into any HSA that is a member of the domain. How this is accomplished and the additional contents of the domain state are detailed in a following section on coordinating domain state, Managing Domain State.

The full domain state is available only on the HSA. The domain state is synchronized between HSA domain members as an exported domain token.

The set of domain keys, *{DK_r}*, always includes one active domain key, and several deactivated domain keys. Domain keys are rotated daily to ensure we comply with Recommendation for Key Management - Part 1 [12]. During domain key rotation, all existing EKTs that contain an HBK encrypted under the outgoing domain key are re-encrypted under the new active domain key. The active key is used to encrypt any new top-level keys, and the expired domain keys can be used only to decrypt previously encrypted exported key tokens for a number of days equivalent to the number of rotated domain keys in the past several days.

Exported Domain Tokens

There is a regular need to synchronize state between domain participants. This is accomplished through exporting the domain state whenever a change is made to the domain. The domain state is exported as an exported domain token.

Field	Description
Name	A domain name to identify this domain.
Members	A list of HSAs that are members of the domain, including their signing and agreement public keys.
Operators	A list of entities, public signing keys, and a role that represents the operators of this service.
Rules	A list of quorum rules for each command that must be satisfied to execute a command on an HSA domain member.

Field	Description
Encrypted Domain Keys	Envelope-encrypted domain keys. The domain keys are encrypted by the signing member for each of the members listed above enveloped to their public agreement key.
Signature	A signature on the domain state produced by HSA host, necessarily a member of the domain that exported the domain state.

The exported domain token forms the fundamental source of trust for entities operating within the domain.

Managing Domain State

The domain state is managed through quorum-authenticated commands. These changes include modifying the list of trusted participants in the domain, modifying the quorum rules for executing HSA commands, and periodically rotating the domain keys. These commands are authenticated on a per-command basis as opposed to authenticated session operations; see the API model depicted in Figure 7.

An HSA, in its initialized and operational state, contains a set of self-generated asymmetric identity keys, a signing key pair, and a key-establishment key pair. Through a manual process a KMS operator can establish an initial domain to be created on a first HSA in a region. This initial domain will consist of a full domain state as defined in Domains and the domain state section, and will be installed through a join command to each of the defined HSA members in the domain.

After an HSA has joined an initial domain, it is bound to the rules defined in that domain. These rules govern the commands that use customer cryptographic keys or make changes to the host or domain state. The authenticated session APIs that use the customer cryptographic keys have been defined earlier.

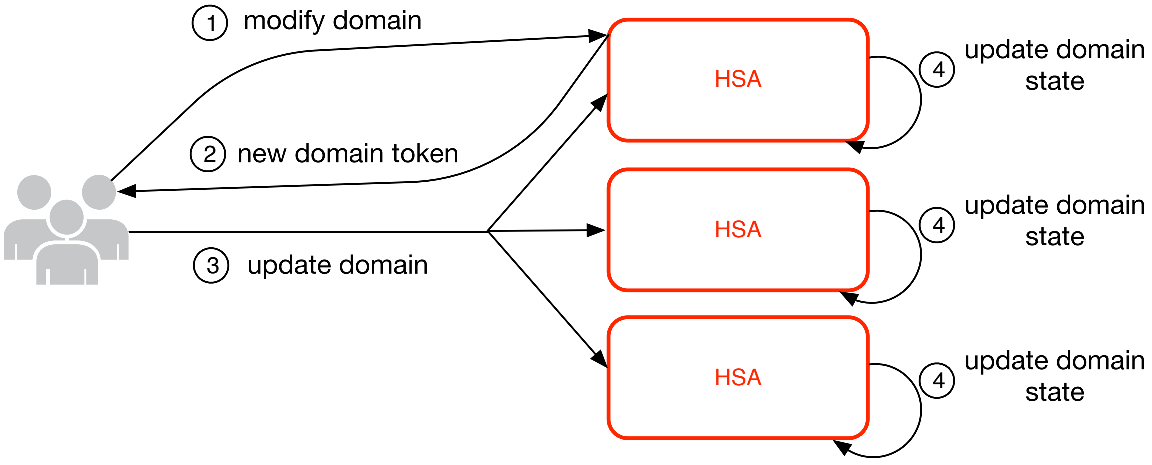

Figure 9: Domain management

Figure 9 depicts how a domain state gets modified. It consists of four steps:

1. A quorum-based command is sent to an HSA to modify the domain.
2. A new domain state is generated and exported as a new exported domain token. The state on the HSA is not modified, meaning that the change is not enacting on the HSA.
3. A second command is sent to each of the HSAs in the newly exported domain token to update their domain state with the new domain token.
4. The HSAs listed in the new exported domain token can authenticate the command and the domain token, and unpack the domain keys to update the domain state on the HSA.

HSAs do not communicate directly with each other. Instead, a quorum of operators requests a change to the domain state resulting in a new exported domain token. A service host member of the domain is used to distribute the new domain state to every HSA in the domain.

The leaving and joining a domain are done through the HSA management functions, and the modification of the domain state is done through the domain management functions.

Command	Description of HSA Management
Leave Domain	Causes an HSA to leave a domain, deleting all remnants and keys of that domain from memory.
Join Domain	Causes an HSA to join a new domain or update its current domain state to the new domain state, using the existing domain as source of the initial set of rules to authenticate this message.

Command	Description of Domain Management
Create Domain	Causes a new domain to be created on an HSA. Returns a first domain token that can be distributed to member HSAs of the domain.
Modify Operators	Adds or removes operators from the list of authorized operators and their roles in the domain.
Modify Members	Adds or removes an HSA from the list of authorized HSAs in the domain.
Modify Rules	Modifies the set of quorum rules required to execute commands on an HSA.
Rotate Domain Keys	Causes a new domain key to be created and marked as the active domain key, moving the existing active key to a deactivated key, and the oldest deactivated key to be removed from the domain state.

Durability Protection

Additional service durability is provided by the use of offline hardware security modules (HSMs), multiple non-volatile storage of exported domain tokens, and redundant storage of CMKs. The offline HSMs are members of the existing domains, and with the exception of not being online and participating in the regular operational fulfillment of domain operations, appear identically in the domain state as the existing HSA members.

The durability design is intended to protect all CMKs in a region should AWS experience a wide-scale loss of either the online HSAs, or the set of CMKs stored within our primary storage system. Imported master keys are not included under the durability protections afforded other CMKs. In the event of a regional-wide failure in AWS KMS, imported master keys may need to be reimported.

The offline HSMs, and the credentials to access them, are stored in safes within monitored safe rooms in multiple independent geographical locations. Each safe requires at least one AWS Security Officer and one AWS KMS Operator, from two independent teams in AWS, to obtain these materials. The use of these materials are governed by internal policy requiring a quorum of AWS KMS Operators to be present.

References

[1] Amazon Web Services, General Reference (Version 1.0), Signing AWS API Request, http://docs.aws.amazon.com/general/latest/gr/signing_aws_api_requests.html.

[2] X9.31-1998: Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA) American National Standards Institute, 1998.

[3] X9.62-2005: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American National Standards Institute, 2005.

[4] X9.63-2011: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, American National Standards Institute, 2011.

[5] Federal Information Processing Standards Publications, FIPS PUB 180-4. Secure Hash Standard, August 2012. Available from http://csrc.nist.gov/publications/fips.

[6] Federal Information Processing Standards Publications, FIPS PUB 186-2. Digital Signature Standard (DSS), January 2000. Available from http://csrc.nist.gov/publications/fips.

[7] Federal Information Processing Standards Publication 197, Announcing the Advanced Encryption Standard (AES) FIPS-186, November 2001. Available from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[8] Federal Information Processing Standards Publication 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008. Available from http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf.

[9] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D, November 2007.

Available from http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[10] Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, NIST Special Publication 800-38E, January 2010. Available from http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf.

[11] Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised), NIST Special Publication 800-56A Revision 2, May 2013. Available from http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf.

[12] Recommendation for Key Management - Part 1: General (Revision 3), NIST Special Publication 800-57A, July 2012, Available from http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf.

[13] PKCS#1 v2.2: RSA Cryptography Standard, RSA Laboratories, October 2012. Available from http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf.

[14] Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS), Brown, D., Turner, S., Internet Engineering Task Force, July 2010, http://tools.ietf.org/html/rfc5753/

[15] SEC 2: Recommended Elliptic Curve Domain Parameters, Standards for Efficient Cryptography Group, Version 2.0, 27 January 2010.

[16] Amazon Web Services, “What is the AWS Encryption SDK,” http://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/introduction.html.

Appendix - Abbreviations and Keys

This section lists abbreviations and keys referenced throughout the document.

Abbreviations

AES	Advanced Encryption Standard
CDK	Customer Data Key
CMK	Customer Master Key
CMKID	Customer Master Key Identifier
ECDH	Elliptic Curve Diffie-Hellman
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	Elliptic Curve Digital Signature Algorithm
EKT	Exported Key Token
GCM	Galois Counter Mode
HBK	HSA Backing Key
HBKID	HSA Backing Key Identifier
HSA	Hardened Security Appliance
RSA	Rivest Shamir and Adleman (cryptologic)
secp384r1	Standards for Efficient Cryptography prime 384-bit random curve 1
SHA256	Secure Hash Algorithm of digest length 256-bits

Keys

Abbreviation	Name: Description
HBK	HSA Backing Key: Hardened Server Appliance Backing Keys are 256-bit master keys, from which specific use keys are derived.
DK	Domain Key: A Domain Key is an AES-256-GCM key. It is shared among all the members of a domain and is used to protect HSA customer keys and HSA-service host session keys.
DKEK	Domain Key Encryption Key: A Domain Key Encryption Key is an AES-256-GCM key generated on a host and used for encrypting the current set of domain keys when sharing of the domain state between HSA hosts.
(dHAK,QHAK)	HSA Agreement Key Pair: Every initiated HSA has a locally generated Elliptic Curve Diffie-Hellman agreement key pair on the curve secp384r1 (NIST-P384).
(dE, QE)	Ephemeral Agreement Key Pair: HSA and service hosts generate ephemeral agreement keys. These are elliptic curve Diffie-Hellman keys on the curve secp384r1 (NIST-P384). These are generated in two use cases, to establish a host-to-host encryption key to transport domain key encryption keys in domain tokens, and to establish HSA-service host session keys to protect sensitive communications.
(dHSK,QHSK)	HSA Signature Key Pair: Every initiated HSA host has a locally generated Elliptic Curve Signature key pair on the curve secp384r1 (NIST-P384).
(dOS,QOS)	Operator Signature Key Pair: Both the service host operators and KMS operators have an identity signing key used to authenticate itself to other domain participants.
SK	Session Key: A session key is created as a result of an authenticated elliptic curve Diffie-Hellman key exchanged between a service host operator and an HSA for the purpose of securing communication between the service host and the member hosts of the domain.

Contributors

The following individuals and organizations contributed to this document:

- Matthew Campagna, Principal Security Engineer, AWS Cryptography
- Ken Beer, Senior Manager Software Development, AWS Cryptography
- Michael Bentkofsky, Software Development Manager, AWS Cryptography
- Sree Pisharody, Senior Product Manager – Technical, AWS Cryptography
- Greg Rubin, Senior Security Engineer, AWS Cryptography
- Aleks Rudzitis, Software Development Engineer, AWS Cryptography

Document Revisions

For the most up to date version of this white paper, please visit:
https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf