AWS Key Management Service Cryptographic Details

Matthew Campagna

May 2015

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 that use AWS. This white paper provides details on the cryptographic operations that are executed within AWS when you use AWS KMS.

Introduction

AWS Key Management Service (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 providing 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 and RESTful APIs to access an elastic, multi-tenet, hardened security appliance (HSA). You will be able to establish your own HSA-based cryptographic contexts under your master keys. These keys are only accessible 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. You can use the AWS KMS console to manage a set of master keys that protect application-specific keys for services in AWS, such as Amazon Elastic Block Store (EBS) volume encryption and Amazon Simple Storage Service (S3) encryption, and to manage developer access to a cloud-based cryptographic service provider to securely obtain, use, and store keys protected by master keys. Access can be managed per master key, per API.

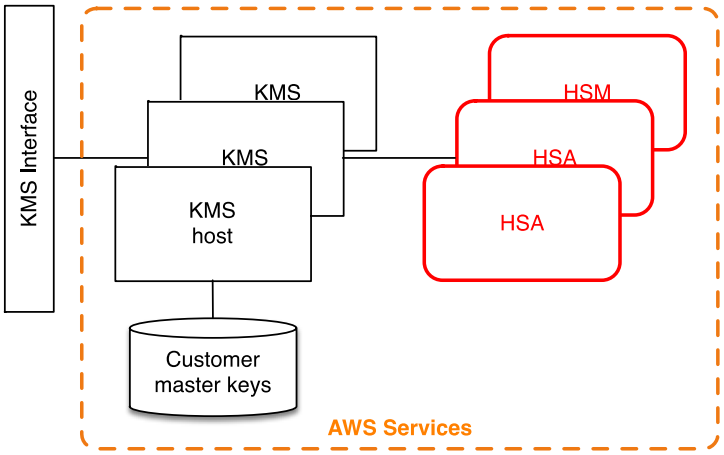

Figure 1: AWS KMS architecture

AWS KMS is a tiered service consisting of web-facing KMS hosts and a tier of HSAs. The grouping of these tiered hosts forms a logical entity called a *domain*. All customer requests to AWS KMS are made over the Security Sockets Layer protocol (SSL) and terminate on a KMS host. The AWS KMS hosts use protocols and procedures defined within this white paper to fulfill those requests through the HSAs. AWS KMS authorizes customer requests through our existing AWS authentication schemes from 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 customer master keys. Confidentiality of your cryptographic keys is crucial.

Access control Use of keys is protected by existing access control policies defined by IAM.

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 and identify anomalous usage, including use of keys by AWS services on the customer's behalf.

In order to achieve these goals the AWS KMS system includes a set of operators that administer "domains." A domain is a regionally defined set of AWS KMS servers, HSAs and human 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 in order to protect HSA-to-HSA communications.

This white paper illustrates how the AWS KMS protects your encryption keys and other data you wish 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 Key Management Service (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.

Encryption

All Encrypt commands used within the HSA and referenced in this document refer to 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.

Digital Signatures

There are two digital signature schemes utilized in AWS KMS: the elliptic curve digital signature algorithm (ECDSA) and RSA. All service 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].

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 (Revised) [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.

Basic Concepts

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

Customer master key: A customer master key is a logical key that represents the top of a customer's key hierarchy. A customer master key 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 customer master key. The alias can be used interchangeably with *keyID* in the AWS KMS APIs.

Permissions: Attached to a customer master key is an IAM policy that defines permissions on the key. The default policy enables IAM users with AWS KMS permissions as well as the root account to manage it.

Grants: In addition to traditional IAM policies, AWS KMS defines grants. Grants are intended to allow delegated use of customer master keys when the duration of usage is not known at the outset. Other AWS services can call KMS on behalf of a customer for an operation that is already permissible for the originating caller.

Data keys: AWS KMS allows authorized entities to obtain data keys protected by a customer master key. They can be returned both in plaintext (un-encrypted) 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 customer master key to use in the decryption process.

Encryption context: *Encryption context* is 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 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 audit 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 an audit will be made that includes "key name": "satellite uplink key."

Customer's Key Hierarchy

The customer's key hierarchy starts with a top-level logical key, called a customer master key (CMK). A CMK represents a container for top-level keys and is uniquely defined within the AWS service namespace with an ARN. The ARN will include a uniquely generated key identifier or customer master 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 HSA backing key 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 HSA backing keys as exported key tokens (EKT).

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

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

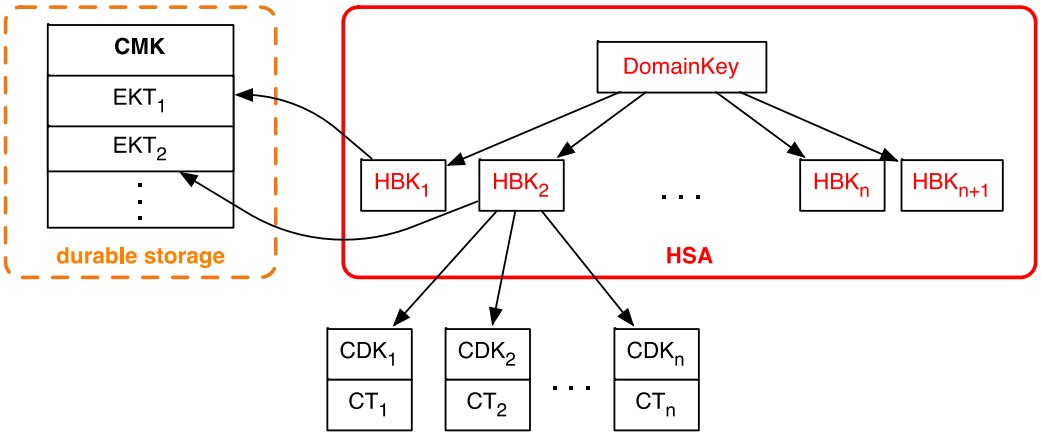

Figure 2: Customer master key hierarchy

You can make requests through AWS KMS to use your customer master keys 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 customer master key (either customer-supplied data or HSA-generated) can only be decrypted 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 SSL connection to AWS KMS. This includes AWS calls 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
Domain Key	A 256-bit AES-GCM key only in memory of HSA used to wrap HSA Backing Keys.	Rotated daily
HSA Backing Key	A 256-bit key only in memory of HSA used to protect customer data keys. Stored encrypted under Domain Keys	Rotated yearly (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 SSL channel.	Rotation and use controlled by application.

Use Cases

This white paper presents two use cases. The first demonstrates how AWS Key Management Service (AWS KMS) performs server-side encryption with customer master keys (CMKs) on an Amazon Elastic Block Storage 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 Elastic Block Storage (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 customer master key in your account. For Amazon EBS to encrypt a volume for you, it must have access to 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 data keys. Now, Amazon EBS will use the CMK to generate AWS KMS-encrypted volume keys.

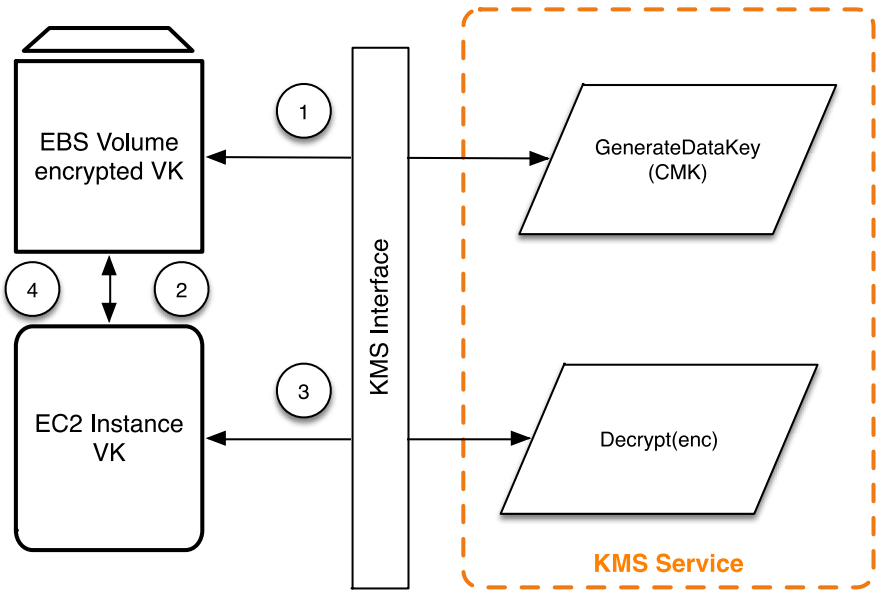

Figure 3: Amazon EBS volume encryption with AWS KMS keys

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

1. Amazon EBS obtains an encrypted volume key under a customer master key through AWS KMS, 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 SSL 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 volume key, and will return the volume key back to the customer over the SSL session.
4. The volume key is stored in memory and 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 SDK includes an API for AWS KMS. Client applications can perform envelope encryption using AWS KMS to encrypt data keys.

```
GenerateDataKeyRequest dataKeyRequest = new GenerateDataKeyRequest();
dataKeyRequest.setKeyId(strKeyId);
dataKeyRequest.setKeySpec("AES_128");
// call AWS KMS to generate a cryptographic key
GenerateDataKeyResult dataKeyResult =
kmsClient.generateDataKey(dataKeyRequest);
// parse out the result
ByteBuffer plaintextKey = dataKeyResult.getPlaintext();
ByteBuffer encryptedKey = dataKeyResult.getCiphertextBlob();
```

The client application can execute the following steps:

1. A request that is made under a customer master key for a new data key. An encrypted data key and a plaintext version of the data key are returned.
2. Within the client application the plaintext data key is used to encrypt the message, the plaintext data key should be deleted from memory.
3. The encrypted data key and encrypted message can be combined into a single object.

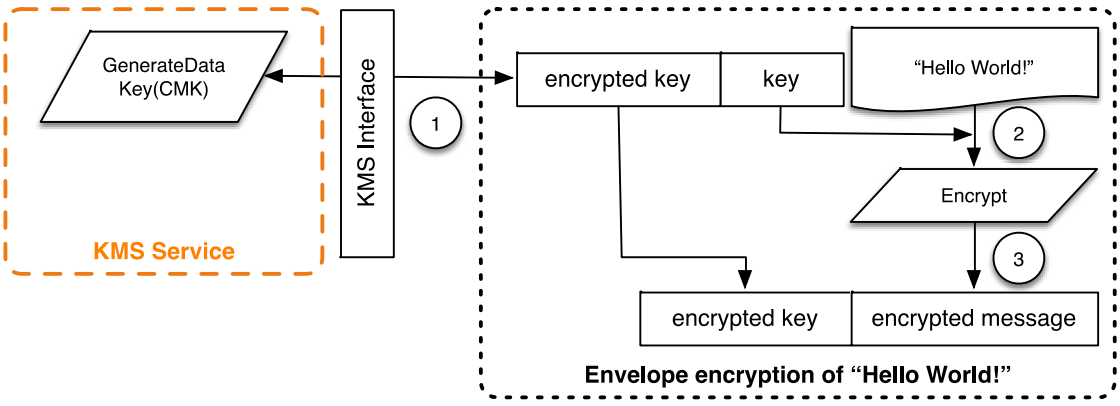

Figure 4: AWS SDK envelope encryption with AWS KMS

At a later time, the envelope-encrypted message can be decrypted using the decrypt functionality to obtain the originally encrypted message.

```
DecryptRequest decryptRequest = new DecryptRequest();
decryptRequest.setCiphertextBlob(encryptedKey);
// call AWS KMS to decrypt the key
DecryptResult decryptResult = kmsClient.decrypt(decryptRequest);
// parse out the decryption result
String strKeyId = decryptResult.getKeyId();
 ByteBuffer plaintextKey = decryptResult.getPlaintext();
```

1. 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 client application will receive the plaintext data key from AWS KMS.
3. The data key can then be used to decrypt the message, returning the initial plaintext.

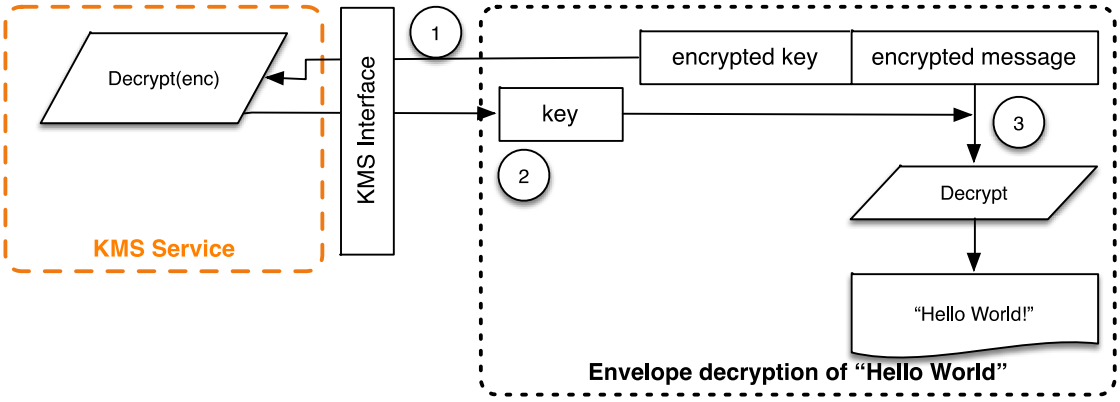

Figure 5: AWS SDK envelope decryption using AWS KMS

Customer Master Keys

A customer master key (CMK) refers to a logical key that may refer to one or more HSA backing keys (HBKs). It is generated as a result of a call to the CreateKey API.

CreateKey request syntax

```
{
        "Description": "string",
        "KeyUsage": "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 Policy: Policy to be attached 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.

This function will accept a key description and a policy. 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>
```

After the ARN is created, a request to an HSA is made over an authenticated session to provision an HSA backing key, a 256-bit key that is associated with this customer master keyID. It can only be generated on an HSA and is designed to never be exported outside of the HSA boundary in plaintext. An HBK is generated on the HSA and exported encrypted under the current domain key DK_0. We refer to these encrypted HSA backing keys as exported key tokens (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 customer master keys and the subsequent HSA backing keys, authorization policies set on your customer master keys, and the cryptographic protections on your associated HSA backing keys. The remaining sections deal with 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 customer master key by creating an alias for the key. Once an alias has been associated with a customer master key, the alias can be used in place of the ARN.

There are multiple levels of authorizations around the use of customer master keys. AWS KMS enables separate authorization policies between the encrypted content and the customer master key. For instance, an AWS KMS envelope-encrypted 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 customer master key.

Customer master keys are given the following default policy:

```
{
```

```
        "Statement":[{
                "Sid":"",
```

```
                "Effect":"Allow",
                "Principal":{ "AWS":"<callerArn>" },
                "Action": "kms:*",
                "Resource":"*"
        }]
}
```

In this policy <callerARN> refers to the AWS account making the CreateKey call.

Encrypting Amazon EBS volumes requires Amazon EBS to be capable of decrypting the customer's volume encryption keys for the duration that the volume is attached to an instance. If the instance reboots or migrates from one instance to another, Amazon EBS needs to decrypt the customer's volume key again, rather than keeping it under an AWS-owned key for the duration of the attachment.

In addition to the use of traditional IAM policies, AWS KMS offers the use of grants. Grants are intended to allow asynchronous use of customer master keys when the duration of usage is not known up-front. Other AWS services can call AWS KMS for some operation that is already permissible for the originating caller. Additional details on using grants can be found at:

https://docs.aws.amazon.com/kms/latest/developerguide/grants.html.

A Grant to a key specifies who can access the key and under what conditions. Grants are an alternate permission mechanism to key policies. If a grant is absent, access to the key is evaluated based on policies attached to the user. By default, Grants do not expire. Grants can be listed, retired, or revoked. Typically, when you are finished using a Grant, you retire it. When you want to end a Grant immediately, you can revoke it.

```
{
        "Constraints": {
                "EncryptionContextEquals":
                {
                        "string" :
                                "string"
                },
                "EncryptionContextSubset":
                {
                        "string" :
                                "string"
                }
        },
        "GrantTokens": [
                "string"
        ],
        "GranteePrincipal": "string",
        "KeyId": "string",
        "Operations": [
                "string"
        ],
        "RetiringPrincipal": "string"
}
```

The request accepts the following data in JSON format.

Optional Constraints: Specifies the conditions under which the actions specified by the Operations parameter are allowed.

Optional GrantTokens: List of grant tokens.

GranteePrincipal: Principal given permission by the grant to use the key identified by the customer master key identifier parameter.

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

Optional Operations: List of operations permitted by the grant. This can be any combination of one or more of the following values:

Decrypt

Encrypt

GenerateDataKey

GenerateDataKeyWithoutPlaintext

ReEncryptFrom

ReEncryptTo

CreateGrant

Optional RetiringPrincipal: Principal given permission to retire the grant. For more information, see RetireGrant.

The function returns a grant identifier and a grant token.

```
{
        "GrantId": "string",
        "GrantToken": "string"
}
```

A grant token is a string that identifies a grant and which can be used to make a grant take effect immediately. A token contains all of the information necessary to create a grant.

Enable and Disable Key

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

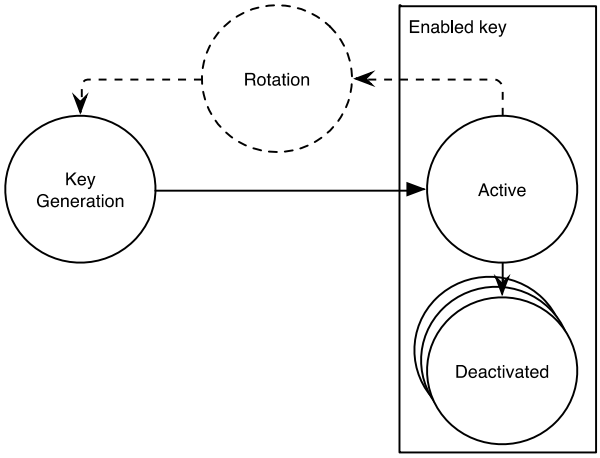

Figure 6: Customer master key states

Rotate Customer Master Key

Customers can induce a rotation of their CMK. The current system allows customers to set a yearly rotation schedule to their customer master key. 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. 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 customer master key (CMK) it can be used to perform other cryptographic operations. Whenever an element is encrypted under a customer master key 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 Key Management Service (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/developerguide/overview.html***

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 be authenticated 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 long-term permissions to generate or use a key.

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

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

AWS KMS, after authenticating the command, will acquire the current active EKT pertaining to the customer master key. 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.
2. Decrypt the *EKT* to obtain the *HBK* = *Decrypt(*DK_i *, EKT).*
3. Determine a 256-bit encryption key *K* from *HBK*.
4. Encrypt the plaintext *CCT* = *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 CCT and the encryption syntax, and the authorization to use the *CMK* the underlying secret cannot be returned.

Depending on the API used, the secret value may also be returned to the customer over the secure channel between the AWS KMS host and the HSA host, and then over the SSL session to the customer from AWS KMS.

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/developerguide/overview.html

Encrypt

A basic function of AWS KMS is to encrypt an object under a customer master key. By design, AWS KMS provides low latency HSA cryptographic operations. 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. To encrypt larger objects it is recommended to use a form of envelope encryption. AWS KMS, after authenticating the command, will acquire the current active EKT pertaining to the customer master key. It will pass the EKT along with the customer-provided plaintext and any encryption context to an HSA 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. Derive a 256-bit encryption key *K* from *HBK*.
3. Encrypt the plaintext *CCT* = *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 CCT 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 CCT and an encryption context. AWS KMS will authenticate the call using AWS signature version 4 signed requests, and extract the HSA backing key identifier for the wrapping key from the ciphertext. The HSA backing key identifier is used to obtain the *EKT* required to decrypt the ciphertext, the Customer Master Key ID, keyId, and the policy for the keyId. The call is authorized based on policies that govern the caller's current user or role, and the policy for the key. The decrypt function is analogous to the encryption function.

Decrypt request syntax

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

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 long-term permissions to perform decryption.

The *CCT*, 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. Derive a 256-bit encryption key *K* from *HBK*.
3. Decrypt the *CCT* to obtain *plaintext* = *Decrypt(K, context, CCT)*.

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 an SSL connection.

Response syntax

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

Re-Encrypting an Encrypted Object

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

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 long-term 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 HSA backing key referenced by the customer ciphertext to the current HSA backing key under the second customer master key. When the customer master keys used in this command are the same, this command moves the customer ciphertext from an old version of an HSA backing key to the latest version of an HSA backing key.

Internal Communication Security

Commands between the service hosts, or operators, and the hardened security appliances (HSAs) are secured through two mechanisms depicted in Figure-7: a quorum-signed request method, and an authenticated session using a host-operator protocol.

The quorum-signed commands are designed so that no solitary operator can obtain HSA-backed keys from the AWS Key Management Service (AWS KMS). 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 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 states including intended or un-intended shutdowns or resets.

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

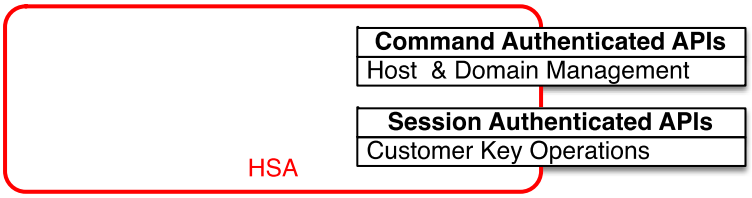

Figure 7: HSA APIs

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 1, and require two members from Role 1 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.

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 data set 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 operator. The exchange is initiated by the service operator 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 operator must re-negotiate a session key.

The customer key operations are executed between the externally facing AWS KMS hosts and the HSAs. These commands affect the creation and use of cryptographic keys and secure random number generation. The commands execute over a session-authenticated channel between the service operators and the HSAs. In addition to the need for authenticity, these sessions require confidentiality. Commands executing over these sessions include the returning of plaintext 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.

An operator 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. This white paper steps you through the HSA - operator session key negotiation.

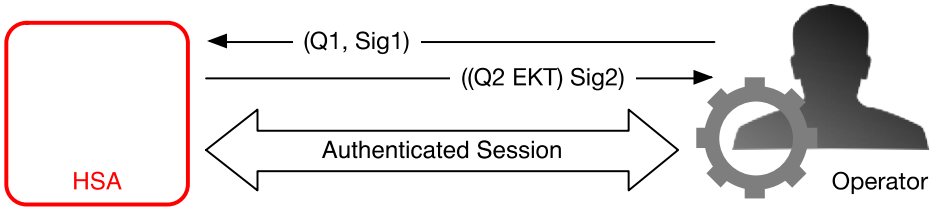

Figure 8: Host-service operator authenticated sessions

The process begins with the service operator 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 operator 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, 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 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 operator verifies the signature on the received key, and completes the ECDH key exchange according to [11], to form a session key *SK*.

During the validity period in the *EKT* the service operator can use the negotiated session key *SK* to send envelope-encrypted commands to the HSA. Every operator-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 Key Management Service (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:

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 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 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 only available 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 a few 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 only be used to decrypt previously encrypted exported key tokens for a limited time.

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 public signing and agreement 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.
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 hardened security appliance (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 an operator can establish an initial domain to be created on a first HSA. This initial domain will consist of a full domain state as defined in Domains and the domain state, 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.

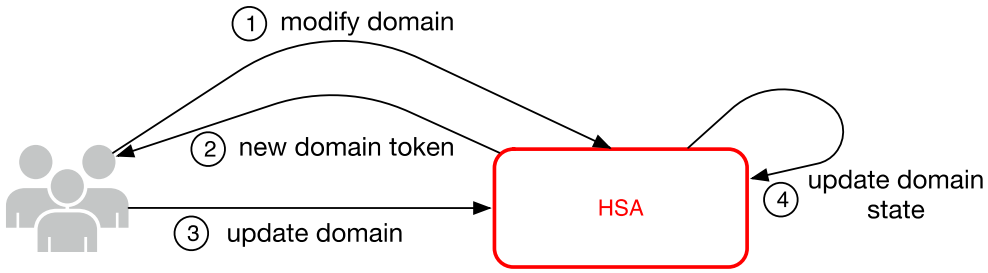

Figure 9: Domain management

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 an exported domain token. The state on the HSA is not modified, meaning that the change is not enacting on the HSA.
3. A second quorum-based command is sent to each of the HSAs in the newly exported domain token to update their domain state with the exported 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.

The leaving and joining a domain is 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
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 customer master keys (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 data in a region should AWS experience a wide-scale loss of either the online hardened security appliances (HSAs), or the set of customer master keys stored within our primary storage system.

The HSM, and the credentials to access them, are stored in safes within monitored safe rooms. Each safe requires at least one AWS KMS safe operator and one AWS KMS system operator to obtain these materials

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, March 2007. Available from http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.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] Public-Key Cryptography Standard (PKCS) #1, RSA Cryptography Standard, RSA Laboratories, October 2012. Available from http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wppdf.

[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.

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 host-operator 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

Abbreviation	**Name:** Description
	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 a host-operator session keys to protect sensitive host-operator 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 operators and human 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 operator and an HSA for the purpose of securing communication between the service operator and the member hosts of the domain.

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