AWS Key Management Service Best Practices

April 2017

Contents

Introduction 1

Identity and Access Management 1

- AWS KMS and IAM Policies 2
- Key Policies 2
- Cross Account Sharing of Keys 4
- CMK Grants 5
- Encryption Context 5
- Multi-Factor Authentication 6

Detective Controls 7

- CMK Auditing 7
- CMK Use Validation 8

Infrastructure Security 8

- Customer Master Keys 8
- Using AWS KMS at Scale 11

Data Protection 11

- Common AWS KMS Use Cases 11
- Enforcing Data at Rest Encryption within AWS Services 13

Incident Response 15

- Security Automation of AWS KMS 15
- Deleting and Disabling CMKs 15

Conclusion 16

Contributors 16

Document Revisions 17

Abstract

AWS Key Management Service (AWS KMS) is a managed service that allows you to concentrate on the cryptographic needs of your applications while Amazon Web Services (AWS) manages availability, physical security, logical access control, and maintenance of the underlying infrastructure. Further, AWS KMS allows you to audit usage of your keys by providing logs of all API calls made on them to help you meet compliance and regulatory requirements.

Customers want to know how to effectively implement AWS KMS in their environment. This whitepaper discusses how to use AWS KMS for each capability described in the AWS Cloud Adoption Framework (CAF) Security Perspective whitepaper, including the differences between the different types of customer master keys, using AWS KMS key policies to ensure least privilege, auditing the use of the keys, and listing some use cases that work to protect sensitive information within AWS.

Introduction

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. AWS KMS uses Hardware Security Modules (HSMs) to protect the security of your keys.[1] You can use AWS KMS to protect your data in AWS services and in your applications. The AWS Key Management Service Cryptographic Details whitepaper describes the design and controls implemented within the service to ensure the security and privacy of your data.[2]

The AWS Cloud Adoption Framework (CAF) whitepaper provides guidance for coordinating the different parts of organizations that are moving to cloud computing.[3] The AWS CAF guidance is broken into areas of focus that are relevant to implementing cloud-based IT systems, which we refer to as *perspectives*. The CAF Security Perspective whitepaper organizes the principles that will help drive the transformation of your organization's security through five core capabilities: Identity and Access Management, Detective Control, Infrastructure Security, Data Protection, and Incident Response.[4]

For each capability in the CAF Security Perspective, this whitepaper provides details on how your organization should use AWS KMS to protect sensitive information across a number of different use cases and the means of measuring progress:

- **Identity and Access Management:** Enables you to create multiple access control mechanisms and manage the permissions for each.
- **Detective Controls:** Provides you the capability for native logging and visibility into the service.
- **Infrastructure Security:** Provides you with the capability to shape your security controls to fit your requirements.
- **Data Protection:** Provides you with the capability for maintaining visibility and control over data.
- **Incident Response:** Provides you with the capability to respond to, manage, reduce harm, and restore operations during and after an incident.

Identity and Access Management

The Identity and Access Management capability provides guidance on determining the controls for access management within AWS KMS to secure your infrastructure according to established best practices and internal policies.

AWS KMS and IAM Policies

You can use AWS Identity and Access Management (IAM) policies in combination with key policies to control access to your customer master keys (CMKs) in AWS KMS. This section discusses using IAM in the context of AWS KMS. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see the AWS IAM User Guide.[5]

Policies attached to IAM identities (that is, users, groups, and roles) are called *identity-based policies* (or *IAM policies*). Policies attached to resources outside of IAM are called *resource-based policies*. In AWS KMS, you must attach resource-based policies to your customer master keys (CMKs). These are called *key policies*. All KMS CMKs have a key policy, and you must use it to control access to a CMK. IAM policies by themselves are not sufficient to allow access to a CMK, although you can use them in combination with a CMK key policy. To do so, ensure that the CMK key policy includes the policy statement that enables IAM policies.[6]

By using an identity-based IAM policy, you can enforce least privilege by granting granular access to KMS API calls within an AWS account. Remember, IAM policies are based on a policy of default-denied unless you explicitly grant permission to a principal to perform an action.

Key Policies

Key policies are the primary way to control access to CMKs in AWS KMS. Each CMK has a key policy attached to it that defines permissions on the use and management of the key. The default policy enables any principals you define, as well as enables the root user in the account to add IAM policies that reference the key. We recommend that you edit the default CMK policy to align with your organization's best practices for least privilege. To access an encrypted resource, the principal needs to have permissions to use the resource, as well as to use the encryption key that protects the resource. If the principal does not have the necessary permissions for either of those actions, the request to use the encrypted resource will be denied.

It's also possible to constrain a CMK so that it can only be used by specific AWS services through the use of the `kms:ViaService` conditional statement within the CMK key policy. For more information, see the AWS KMS Developer Guide.[7]

To create and use an encrypted Amazon Elastic Block Store (EBS) volume, you need permissions to use Amazon EBS. The key policy associated with the CMK would need to include something similar to the following:

```
{
   "Sid": "Allow for use of this Key",
   "Effect": "Allow",
   "Principal": {
     "AWS": "arn:aws:iam::111122223333:role/UserRole"
   },
   "Action": [
     "kms:GenerateDataKeyWithoutPlaintext",
     "kms:Decrypt"
   ],
   "Resource": "*"
 },
 {
   "Sid": "Allow for EC2 Use",
   "Effect": "Allow",
   "Principal": {
     "AWS": "arn:aws:iam::111122223333:role/UserRole"
   },
   "Action": [
     "kms:CreateGrant",
     "kms:ListGrants",
     "kms:RevokeGrant"
   ],
   "Resource": "*",
   "Condition": {
     "StringEquals": {
       "kms:ViaService": "ec2.us-west-2.amazonaws.com"
     }
   }
 }
```

In this CMK policy, the first statement provides a specified IAM principal the ability to generate a data key and decrypt that data key from the CMK when necessary. These two APIs are necessary to encrypt the EBS volume while it's attached to an Amazon Elastic Compute Cloud (EC2) instance.

The second statement in this policy provides the specified IAM principal the ability to create, list, and revoke grants for Amazon EC2. Grants are used to delegate a subset of permissions to AWS services, or other principals, so that they can use your keys on your behalf. In this case, the condition policy explicitly ensures that only Amazon EC2 can use the grants. Amazon EC2 will use them to re-attach an encrypted EBS volume back to an instance if the volume gets detached due to a planned or unplanned outage. These events will be recorded within AWS CloudTrail when, and if, they do occur for your auditing.

When developing a CMK policy, you should keep in mind how policy statements are evaluated within AWS. This means that if you have enabled IAM to help control access to a CMK, when

AWS evaluates whether a permitted action is to be allowed or denied, the CMK policy is joined with the IAM policy. Additionally, you should ensure that the use and management of a key is restricted to the parties that are necessary.

Least Privilege / Separation of Duties

Key policies specify a resource, action, effect, principal, and conditions to grant access to CMKs. Key policies allow you to push more granular permissions to CMKs to enforce least privilege. For example, an application might make a KMS API call to encrypt data but there is no use case for that same application to decrypt data. In that use case, a key policy could grant access to the *kms:Encrypt* action but not *kms:Decrypt* and reduce the possibility for exposure. Additionally, AWS allows you to separate the usage permissions from administration permissions associated with the key. This means that an individual may have the ability to manipulate the key policy, but might not have the necessary permissions to use the key for cryptographic functions.

Given that your CMKs are being used to protect your sensitive information, you should work to ensure that the corresponding key policies follow a model of least privilege. This includes ensuring that you do **NOT** include kms:* permissions in an IAM policy. This policy would grant the principal both administrative and usage permissions on all CMKs to which the principal has access. Similarly, including kms:* permissions for the principals within your key policy gives them both administrative and usage permissions on the CMK.

It's important to remember that explicit deny policies take precedence over implicit deny policies. When you use NotPrincipal in the same policy statement as "Effect: Deny", the permissions specified in the policy statement are explicitly denied to all principals *except* for the ones specified. A top-level KMS policy can explicitly deny access to virtually all KMS operations except for the roles that actually need them. This technique helps prevent unauthorized users from granting themselves KMS access.

Cross Account Sharing of Keys

Delegation of permissions to a CMK within AWS KMS can occur when you include the root principal of a trusted account within the CMK key policy. The trusted account then has the ability to further delegate these permissions to IAM users and roles within their own account using IAM policies. While this approach may simplify the management of the key policy, it also relies on the trusted accounts to ensure that the delegated permissions are correctly managed. The other approach would be to explicitly manage permissions to all authorized users using only the KMS key policy, which, in turn, could make the key policy complex and less manageable. Regardless of the approach you take, the specific trust should be broken out on a per key basis to ensure that you adhere to the least privilege model.

CMK Grants

Key policy changes follow the same permissions model used for policy editing elsewhere in AWS. That is, users either have permission to change the key policy or they do not. Users with the `PutKeyPolicy` permission for a CMK can completely replace the key policy for a CMK with a different key policy of their choice. You can use key policies to allow other principals to access a CMK, but key policies work best for relatively static assignments of permissions. To enable more granular permissions management, you can use grants. Grants are useful when you want to define scoped-down, temporary permissions for other principals to use your CMK on your behalf in the absence of a direct API call from you.

It's important to be aware of the grants per key and grants for a principal per key limits when you design applications that use grants to control access to keys. Ensure that the retiring principal retires a grant after it's used to avoid hitting these limits.

Encryption Context

In addition to limiting permission to the AWS KMS APIs, AWS KMS also gives you the ability to add an additional layer of authentication for your KMS API calls utilizing encryption context. The encryption context is a key-value pair of additional data that you want associated with AWS KMS-protected information. This is then incorporated into the additional authenticated data (AAD) of the authenticated encryption in AWS KMS-encrypted ciphertexts. If you submit the encryption context value in the encryption operation, you are required to pass it in the corresponding decryption operation. You can use the encryption context inside your policies to enforce tighter controls for your encrypted resources. Because the encryption context is logged in CloudTrail, you can get more insight into the usage of your keys from an audit perspective. Be aware that the encryption context is not encrypted and will be visible within CloudTrail logs. The encryption context should not be considered sensitive information and should not require secrecy.

AWS services that use AWS KMS use encryption context to limit the scope of keys. For example, Amazon EBS sends the volume ID as the encryption context when encrypting/decrypting a volume, and when you take a snapshot the snapshot ID is used as the context. If Amazon EBS did not use this encryption context, an EC2 instance would be able to decrypt any EBS volume under that specific CMK.

An encryption context can also be used for custom applications that you develop, and acts as an additional layer of control by ensuring that decrypt calls will succeed only if the encryption context matches what was passed in the encrypt call. If the encryption context for a specific application does not change, you can include that context within the AWS KMS key policy as a conditional statement. For example, if you have an application that requires the ability to encrypt and decrypt data, you can create a key policy on the CMK that ensures that it provides expected values. In the following policy, it is checking that the application name "ExampleApp" and its current version "1.0.24" are the values that are passed to AWS KMS during the encrypt and decrypt calls. If different values are passed, the call will be denied and the decrypt or encrypt action will not be performed.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp",
      "kms:EncryptionContext:Version": "1.0.24"
    }
  }
}
```

This use of encryption context will help to further ensure that only authorized parties and/or applications can access and use the CMKs. Now the party will need to have IAM permissions to AWS KMS, a CMK policy that allows them to use the key in the requested fashion, and finally know the expected encryption context values.

Multi-Factor Authentication

To provide an additional layer of security over specific actions, you can implement an additional layer of protection using multi-factor authentication (MFA) on critical KMS API calls. Some of those calls are `PutKeyPolicy`, `ScheduleKeyDeletion`, `DeleteAlias`, and `DeleteImportedKeyMaterial`. This can be accomplished through a conditional statement within the key policy that checks for when or if an MFA device was used as part of authentication.

If someone attempts to perform one of the critical AWS KMS actions, the following CMK policy will validate that their MFA was authenticated within the last 300 seconds, or 5 minutes, before performing the action.

```
{
  "Sid": "MFACriticalKMSEvents",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": [
    "kms:DeleteAlias",
    "kms:DeleteImportedKeyMaterial",
    "kms:PutKeyPolicy",
    "kms:ScheduleKeyDeletion"
  ],
  "Resource": "*",
  "Condition":{
    " NumericLessThan ":{"aws: MultiFactorAuthAge":"300"}
  }
}
```

Detective Controls

The Detective Controls capability ensures that you properly configure AWS KMS to log the necessary information you need to gain greater visibility into your environment.

CMK Auditing

AWS KMS is integrated with CloudTrail. To audit the usage of your keys in AWS KMS, you should enable CloudTrail logging in your AWS account. This ensures that all KMS API calls made on keys in your AWS account are automatically logged in files that are then delivered to an Amazon Simple Storage Service (S3) bucket that you specify. Using the information collected by CloudTrail, you can determine what request was made, the source IP address from which the request was made, who made the request, when it was made, and so on.

AWS KMS integrates natively with many other AWS services to make monitoring easy. You can use these AWS services, or your existing security tool suite, to monitor your CloudTrail logs for specific actions such as `ScheduleKeyDeletion`, `PutKeyPolicy`, `DeleteAlias`, `DisableKey`, `DeleteImportedKeyMaterial` on your KMS key. Furthermore, AWS KMS emits Amazon CloudWatch Events when your CMK is rotated, deleted, and imported key material in your CMK expires.

CMK Use Validation

In addition to capturing audit data associated with key management and use, you should ensure that the data you are reviewing aligns with your established best practices and policies. One method is to continuously monitor and verify the CloudTrail logs as they come in. Another method is to use AWS Config rules. By using AWS Config rules you can ensure that the configuration of many of the AWS services are set up appropriately. For example, with EBS volumes you can use the AWS Config rule `ENCRYPTED_VOLUMES` to validate that attached EBS volumes are encrypted.

Key Tags

A CMK can have a tag applied to it for a variety of purposes. The most common use is to correlate a specific CMK back to a business category (such as a cost center, application name, or owner). The tags can then be used to verify that the correct CMK is being used for a given action. For example, in CloudTrail logs, for a given KMS action you can verify that the CMK being used belongs to the same business category as the resource that it's being used on. Previously, this might have required a look up within a resource catalog, but now this external lookup is not required because of tagging within AWS KMS as well as many of the other AWS services.

Infrastructure Security

The Infrastructure Security capability provides you with best practices on how to configure AWS KMS to ensure that you have an agile implementation that can scale with your business while protecting your sensitive information.

Customer Master Keys

Within AWS KMS, your key hierarchy starts with a CMK. A CMK can be used to directly encrypt data blocks up to 4 KB or it can be used to secure data keys, which protect underlying data of any size.

AWS-managed and Customer-managed CMKs

CMKs can be broken down into two general types: AWS-managed and customer-managed. An AWS-managed CMK is created when you choose to enable server-side encryption of an AWS resource under the AWS-managed CMK for that service for the first time (e.g., SSE-KMS). The AWS-managed CMK is unique to your AWS account and the Region in which it's used. An AWS-managed CMK can only be used to protect resources within the specific AWS service for which it's created. It does not provide the level of granular control that a customer-managed CMK provides. For more control, a best practice is to use a customer-managed CMK in all supported AWS services and in your applications. A customer-managed CMK is created at your request and should be configured based upon your explicit use case.

The following chart summarizes the key differences and similarities between AWS-managed CMKs and customer-managed CMKs.

	AWS-managed CMK	Customer-managed CMK
Creation	AWS generated on customer's behalf	Customer generated
Rotation	Once every three years automatically	Once a year automatically through opt-in or on-demand manually
Deletion	Can't be deleted	Can be deleted
Scope of use	Limited to a specific AWS service	Controlled via KMS/IAM policy
Key Access Policy	AWS managed	Customer managed
User Access Management	IAM policy	IAM policy

For customer-managed CMKs, you have two options for creating the underlying key material. When you choose to create a CMK using AWS KMS, you can let KMS create the cryptographic material for you, or you can choose to import your own key material. Both of these options provide you with the same level of control and auditing for the use of the CMK within your environment. The ability to import your own cryptographic material allows you to do the following:

- Prove that you generated the key material using your approved source that meets your randomness requirements.
- Use key material from your own infrastructure with AWS services, and use AWS KMS to manage the lifecycle of that key material within AWS.
- Gain the ability to set an expiration time for the key material in AWS and manually delete it, but also make it available again in the future.
- Own the original copy of the key material, and to keep it outside of AWS for additional durability and disaster recovery during the complete lifecycle of the key material.

The decision to use imported key material or KMS-generated key material would depend on your organization's policies and compliance requirements.

Key Creation and Management

Since AWS makes creating and managing keys easy through the use of AWS KMS, we recommend that you have a plan for how to use the service to best control the blast radius around individual keys. Previously, you may have used the same key across different geographic regions, environments, or even applications. With AWS KMS, you should define data classification levels and have at least one CMK per level. For example, you could define a CMK for data classified as "Confidential," and so on. This ensures that authorized users only have permissions for the key material that they require to complete their job.

You should also decide how you want to manage usage of AWS KMS. Creating KMS keys within each account that requires the ability to encrypt and decrypt sensitive data works best for most customers, but another option is to share the CMKs from a few centralized accounts. Maintaining the CMKs in the same account as the majority of the infrastructure using them helps users provision and run AWS services that use those keys. AWS services don't allow for cross-account searching unless the principal doing the searching has explicit List* permissions on resources owned by the external account. This can also only be accomplished via the CLI or SDK, and not through service console-based searches. Additionally, by storing the credentials in the local accounts, it might be easier to delegate permissions to individuals who know the IAM principals that require access to the specific CMKs. If you were sharing the keys via a centralized model, the AWS KMS administrators would need to know the full Amazon Resource Name (ARN) for all users of the CMKs to ensure least privilege. Otherwise, the administrators might provide overly permissive permissions on the keys.

Your organization should also consider the frequency of rotation for CMKs. Many organizations rotate CMKs yearly. For customer-managed CMKs with KMS-generated key material, this is easy to enforce. You simply have to opt in to a yearly rotation schedule for your CMK. When the CMK is due for rotation, a new backing key is created and marked as the active key for all new requests to protect information. The old backing key remains available for use to decrypt any existing ciphertext values that were encrypted using this key. To rotate CMKs more frequently, you can also call `UpdateAlias` to point an alias to a new CMK, as described in the next section. The `UpdateAlias` method works for both customer-managed CMKs and CMKs with imported key material. AWS has found that the frequency of key rotation is highly dependent upon laws, regulations, and corporate policies.

Key Aliases

A key alias allows you to abstract key users away from the underlying Region-specific key ID and key ARN. Authorized individuals can create a key alias that allows their applications to use a specific CMK independent of the Region or rotation schedule. Thus, multi-Region applications can use the same key alias to refer to KMS keys in multiple Regions without worrying about the key ID or the key ARN. You can also trigger manual rotation of a CMK by pointing a given key alias to a different CMK. Similar to how Domain Name Services (DNS) allows the abstraction of IP addresses, a key alias does the same for the key ID. When you are creating a key alias, we recommend that you determine a naming scheme that can be applied across your accounts such as *alias/<Environment>-<Function>-<Service Team>*.

It should be noted that CMK aliases can't be used within policies. This is because the mapping of aliases to keys can be manipulated outside the policy, which would allow for an escalation of privilege. Therefore, key IDs must be used in KMS key policies, IAM policies, and KMS grants.

Using AWS KMS at Scale

As noted earlier, a best practice is to use at least one CMK for a particular class of data. This will help you define policies that scope down permissions to the key and hence the data to authorized users. You may choose to further distribute your data across multiple CMKs to provide stronger security controls within a given data classification.

AWS recommends using envelope encryption to scale your KMS implementation. Envelope encryption is the practice of encrypting plaintext data with a unique data key, and then encrypting the data key with a key encryption key (KEK). Within AWS KMS, the CMK is the KEK. You can encrypt your message with the data key and then encrypt the data key with the CMK. Then the encrypted data key can be stored along with the encrypted message. You can cache the plaintext version of the data key for repeated use, reducing the number of requests to AWS KMS. Additionally, envelope encryption can help to design your application for disaster recovery. You can move your encrypted data as-is between Regions and only have to re-encrypt the data keys with the Region-specific CMKs.

The AWS Cryptographic team has released an AWS Encryption SDK that makes it easier to use AWS KMS in an efficient manner. This SDK transparently implements the low-level details for using AWS KMS. It also provides developers options for protecting their data keys after use to ensure that the performance of their application isn't significantly affected by encrypting your sensitive data.

Data Protection

The Data Protection capability addresses some of the common AWS use cases for using AWS KMS within your organization to protect your sensitive information.

Common AWS KMS Use Cases

Encrypting PCI Data Using AWS KMS

Since security and quality controls in AWS KMS have been validated and certified to meet the requirements of PCI DSS Level 1 certification, you can directly encrypt Primary Account Number (PAN) data with an AWS KMS CMK. The use of a CMK to directly encrypt data removes some of the burden of managing encryption libraries. Additionally, a CMK can't be exported from AWS KMS, which alleviates the concern about the encryption key being stored in an insecure manner. As all KMS requests are logged in CloudTrail, use of the CMK can be audited by reviewing the CloudTrail logs. It's important to be aware of the requests per second limit when designing applications that use the CMK directly to protect Payment Card Industry (PCI) data.

Secret Management Using AWS KMS and Amazon S3

Although AWS KMS primarily provides key management functions, you can leverage AWS KMS and Amazon S3 to build your own secret management solution.

Create a new Amazon s3 bucket to hold your secrets. Deploy a bucket policy onto the bucket to limit access to only authorized individuals and services. The secrets stored in the bucket utilize a predefined prefix per file to allow for granular control of access to the secrets. Each secret, when placed in the S3 bucket, is encrypted using a specific customer-managed KMS key. Furthermore, due to the highly sensitive nature of the information being stored within this bucket, S3 access logging or CloudTrail Data Events are enabled for audit purposes. Then, when a user or service requires access to the secret, they assume an identity within AWS that has permissions to use both the object in the S3 bucket as well as the KMS key. An application that runs in an EC2 instance uses an instance role that has the necessary permissions.

Encrypting Lambda Environment Variables

By default, when you create or update Lambda functions that use environment variables, those variables are encrypted using AWS KMS. When your Lambda function is invoked, those values are decrypted and made available to the Lambda code. You have the option to use the default KMS key for Lambda or specify a specific CMK of your choice.

To further protect your environment variables, you should select the “Enable encryption helpers” checkbox. By selecting this option, your environment variables will also be individually encrypted using a CMK of your choice, and then your Lambda function will have to specifically decrypt each encrypted environment variable that is needed.

Encrypting Data within Systems Manager Parameter Store

Amazon EC2 Systems Manager is a collection of capabilities that can help you automate management tasks at scale. To efficiently store and reference sensitive configuration data such as passwords, license keys, and certificates, the Parameter Store lets you protect sensitive information within secure string parameters.

A secure string is any sensitive data that needs to be stored and referenced in a secure manner. If you have data that you don't want users to alter or reference in clear text, such as domain join passwords or license keys, then specify those values using the Secure String data type. You should use secure strings in the following circumstances:

- You want to use data/parameters across AWS services without exposing the values as clear text in commands, functions, agent logs, or CloudTrail logs.
- You want to control who has access to sensitive data.
- You want to be able to audit when sensitive data is accessed using CloudTrail.

- You want AWS-level encryption for your sensitive data and you want to bring your own encryption keys to manage access.

By selecting this option when you create your parameter, the Systems Manager encrypts that value when it's passed into a command and decrypts it when processing it on the managed instance. The encryption is handled by AWS KMS and can be either a default KMS key for the Systems Manager or you can specify a specific CMK per parameter.

Enforcing Data at Rest Encryption within AWS Services

Your organization might require the encryption of all data that meets a specific classification. Depending on the specific service, you can enforce data encryption policies through preventative or detective controls. For some services like Amazon S3, a policy can prevent storing unencrypted data. For other services, the most efficient mechanism is to monitor the creation of storage resources and check whether encryption is enabled appropriately. In the event that unencrypted storage is created, you have a number of possible responses ranging from deleting the storage resource to notifying an administrator.

Data at Rest Encryption with Amazon S3

Using Amazon S3, it's possible to deploy an S3 bucket policy that ensures that all objects being uploaded are encrypted. The policy looks like the following:

```
{
   "Version":"2012-10-17",
   "Id":"PutObjPolicy",
   "Statement":[{
         "Sid":"DenyUnEncryptedObjectUploads",
         "Effect":"Deny",
         "Principal":"*",
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::YourBucket/*",
         "Condition":{
            "StringNotEquals":{
               "s3:x-amz-server-side-encryption":"aws:kms"
            }
         }
      }
   ]
}
```

Note that this doesn't cause objects already in the bucket to be encrypted. This policy denies attempts to add new objects to the bucket unless those objects are encrypted. Objects already in the bucket before this policy is applied will remain either encrypted or unencrypted based on how they were first uploaded.

Data at Rest Encryption with Amazon EBS

You can create Amazon Machine Images (AMIs) that make use of encrypted EBS boot volumes and use the AMIs to launch EC2 instances. The stored data is encrypted, as is the data transfer path between the EBS volume and the EC2 instance. The data is decrypted on the hypervisor of that instance on an as-needed basis, then stored only in memory. This feature aids your security, compliance, and auditing efforts by allowing you to verify that all of the data that you store on the EBS volume is encrypted, whether it's stored on a boot volume or on a data volume. Further, because this feature makes use of AWS KMS, you can track and audit all uses of the encryption keys.

There are two methods to ensure that EBS volumes are always encrypted. You can verify that the encryption flag as part of the `CreateVolume` context is set to "true" through an IAM policy. If the flag is not "true" then the IAM policy can prevent an individual from creating the EBS volume. The other method is to monitor the creation of EBS volumes. If a new EBS volume is created, CloudTrail will log an event. A Lambda function can be triggered by the CloudTrail event to check if the EBS volume is encrypted or not, and also what KMS key was used for the encryption.

An AWS Lambda function can respond to the creation of an unencrypted volume in several different ways. The function could call the `CopyImage` API with the encrypted option to create a new encrypted version of the EBS volume and then attach it to the instance and delete the old version. Some customers choose to automatically delete the EC2 instance that has the unencrypted volume. Others choose to automatically quarantine the instance it by applying security groups that prevent most inbound connections. It's also easy to write a Lambda function that posts to an Amazon Simple Notification Service (SNS) topic that alerts administrators to do a manual investigation and intervention. Note that most enforcement responses can—and should—be accomplished programmatically without human intervention.

Data at Rest Encryption with Amazon RDS

Amazon Relational Database Service (RDS) builds on Amazon EBS encryption to provide full disk encryption for database volumes. When you create an encrypted database instance with Amazon RDS, Amazon RDS creates an encrypted EBS volume on your behalf to store the database. Data stored at rest on the volume, database snapshots, automated backups, and read replicas are all encrypted under the KMS CMK that you specified when you created the database instance.

Similar to Amazon EBS, you can set up an AWS Lambda function to monitor for the creation of new RDS instances via the `CreateDBInstance` API call via CloudTrail. Within the `CreateDBInstance` event, ensure that `KmsKeyId` parameter is set to the expected CMK.

Incident Response

The Incident Response capability focuses on your organization's capability to remediate incidents that may involve AWS KMS.

Security Automation of AWS KMS

During your monitoring of your CMKs, if a specific action is detected, an AWS Lambda function could be configured to disable the CMK or perform any other incident response actions as dictated by your local security policies. Without human intervention, a potential exposure could be cut off in minutes by leveraging the automation tools inside AWS.

Deleting and Disabling CMKs

While deleting CMKs is possible it has significant ramifications to an organization. You should first consider whether it's sufficient to set the CMK state to disabled on keys that you no longer intend to use. This will prevent all future use of the CMK. The CMK is still available, however, and can be re-enabled in the future if it's needed. Disabled keys are still stored by AWS KMS; thus, they continue to incur recurring storage charges. You should strongly consider disabling keys instead of deleting them until you are confident in their encrypted data management.

Deleting a key must be very carefully thought out. Data can't be decrypted if the corresponding CMK has been deleted. Moreover, once a CMK is deleted, it's gone forever. AWS has no means to recover a deleted CMK once it's finally deleted. Just as with other critical operations in AWS, you should apply a policy that requires MFA for CMK deletion.

To help ensure that a CMK is not deleted by mistake, KMS enforces a minimum waiting period of seven days before the CMK is actually deleted. You can choose to increase this waiting period up to a maximum value of 30 days. During the waiting period, the CMK is still stored in KMS in a "Pending Deletion" state. It can't be used for encrypt or decrypt operations. Any attempt to use a key that is in the "Pending Deletion" state for encryption or decryption will be logged to CloudTrail. You can set an Amazon CloudWatch Alarm for these events in your CloudTrail logs. This gives you a chance to cancel the deletion process if needed. Until the waiting period has expired, the CMK can be recovered from the "Pending Deletion" state and restored to either the disabled or enabled state.

Finally, it should also be noted that if you are using a CMK with imported key material, you can delete the imported key material immediately. This is different from deleting a CMK directly in several ways. When you perform the `DeleteImportedKeyMaterial` action, AWS KMS deletes the key material and the CMK key state changes to pending import. When the key material is deleted, the CMK is immediately unusable. There is no waiting period. To enable use of the CMK again, you must reimport the same key material. Deleting key material affects the CMK right away, but data encryption keys that are actively in use by AWS services are not immediately affected.

For example, let's say a CMK using your imported material was used to encrypt an object being placed in an S3 bucket using SSE-KMS.[8] Right before you upload the object into the S3 bucket, you place the imported material into your CMK. After the object is uploaded, you can delete your key material from that CMK. The object will continue to sit in the S3 bucket in an encrypted state, but no one will be able to access it until the same key material is re-imported into the CMK. This flow obviously requires precise automation for importing and deleting key material from a CMK, but can provide an additional level of control within an environment.

Conclusion

AWS KMS provides your organization with a fully managed service to centrally control your encryption keys. Its native integration with other AWS services makes it easier for AWS KMS to encrypt the data that you store and process.

By taking the time to properly architect and implement AWS KMS, you can ensure that your encryption keys are secure and available for applications and their authorized users. Additionally, you can show your auditors detailed logs associated with your key usage.

Contributors

The following individuals and organizations contributed to this document:

- Matthew Bretan, Senior Security Consultant, AWS Professional Services
- Sree Pisharody, Senior Product Manager – Technical, AWS Cryptography
- Ken Beer, Senior Manager Software Development, AWS Cryptography
- Brian Wagner, Security Consultant, AWS Professional Services
- Eugene Yu, Managing Consultant, AWS Professional Services
- Michael St.Onge, Global Cloud Security Architect, AWS Professional Services
- Balaji Palanisamy, Senior Consultant, AWS Professional Services
- Jonathan Rault, Senior Consultant, AWS Professional Services
- Reef Dsouza, Consultant, AWS Professional Services
- Paco Hope, Principal Consultant, AWS Professional Services

Document Revisions

For the most up-to-date version of this white paper, please visit:

https://d0.awsstatic.com/whitepapers/KMS-Best-Practices.pdf

Notes

1 http://docs.aws.amazon.com/kms/latest/developerguide/overview.html

2 https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf

3 https://d0.awsstatic.com/whitepapers/aws_cloud_adoption_framework.pdf

4 https://d0.awsstatic.com/whitepapers/AWS_CAF_Security_Perspective.pdf

5 http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

6 http://docs.aws.amazon.com/kms/latest/developerguide/key-policies.html#key-policy-default-allow-root-enable-iam

7 http://docs.aws.amazon.com/kms/latest/developerguide/policy-conditions.html#conditions-kms-via-service

8 http://docs.aws.amazon.com/kms/latest/developerguide/services-s3.html#sse