OpenSecOps SOAR KMS Keys

Standard Operating Procedure

Table of Contents

1 Introduction

The purpose of this document is to specify baseline settings for AWS KMS keys so that teams can provision encryption keys in a safe, conformant way.

2 Scope of SOP

This document covers AWS KMS keys and how they are to be set up in the OpenSecOps organisation.

Document Versions

Version	Date	Changes	Author
1.0	2022-09-12	First version	Peter Bengtson
1.1	2025-04-07	Replaced "Delegat" with "OpenSecOps" throughout	Peter Bengtson

Please add new entries *above* already existing rows.

3 Enable Automatic Key Rotation

KMS keys **MUST** have automatic key rotation enabled. Failure to enable automatic key rotation generates a `MEDIUM` security incident.

When you enable automatic key rotation for a customer-managed CMK, AWS KMS generates new cryptographic material for the CMK every year. AWS KMS also saves the CMK's older cryptographic material in perpetuity so it can be used to decrypt data that it encrypted. AWS KMS does not delete any rotated key material until you delete the CMK.

For full details, see
https://docs.aws.amazon.com/kms/latest/developerguide/rotate-keys.html

3.1 CloudFormation

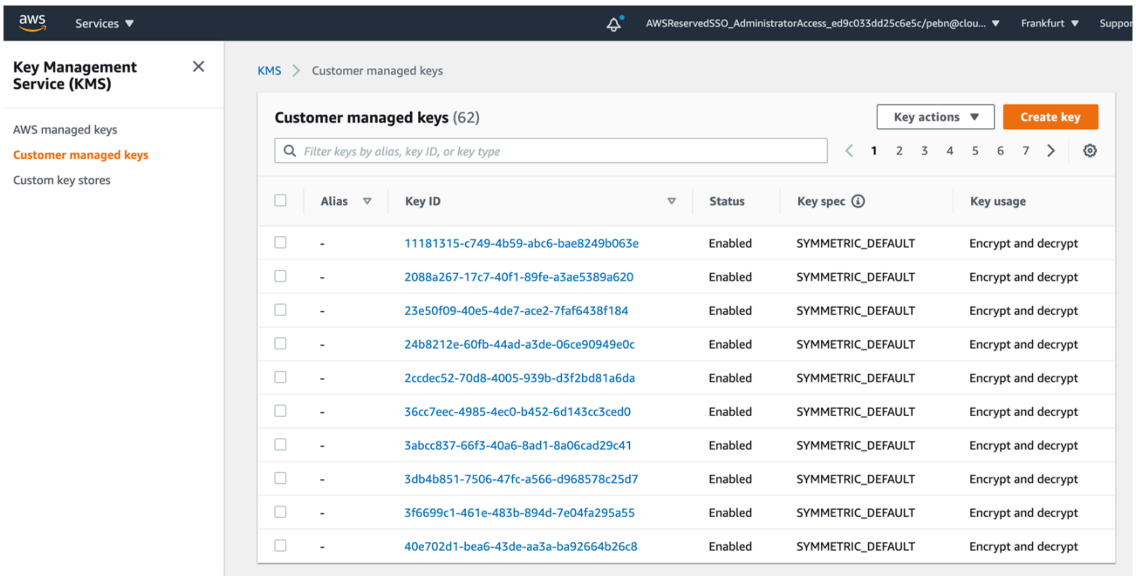
```
ExampleKey:
  Type: AWS::KMS::Key
  Properties:
    Description: "For encrypting the foo-bar SNS topic"
    EnableKeyRotation: true
```

3.2 Terraform

```
resource "aws_kms_key" "a" {
  description         = "For encrypting the foo-bar SNS topic"
  enable_key_rotation = true
}
```

3.3 Console

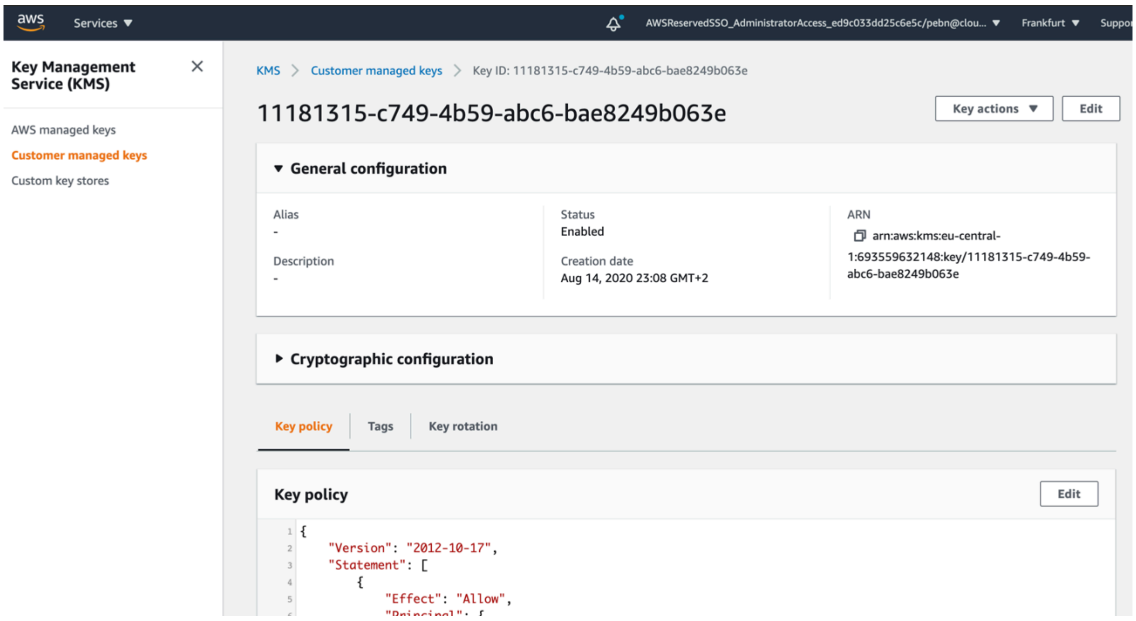
First, navigate to the KMS console page:


aws
Services ▼
AWSReservedSSO_AdministratorAccess_ed9c033dd25c6e5c/pebn@clou... ▼
Frankfurt ▼
Suppor
Key Management Service (KMS)
AWS managed keys
Customer managed keys
Custom key stores
KMS > Customer managed keys
Key actions ▼
Create key
Filter keys by alias, key ID, or key type
< 1 2 3 4 5 6 7 >

Customer managed keys (62)

	Alias	Key ID	Status	Key spec	Key usage
☐	-	11181315-c749-4b59-abc6-bae8249b063e	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	2088a267-17c7-40f1-89fe-a3ae5389a620	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	23e50f09-40e5-4de7-ace2-7faf6438f184	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	24b8212e-60fb-44ad-a3de-06ce90949e0c	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	2ccdec52-70d8-4005-939b-d3f2bd81a6da	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	36cc7eec-4985-4ec0-b452-6d143cc3ced0	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	3abcc837-66f3-40a6-8ad1-8a06cad29c41	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	3db4b851-7506-47fc-a566-d968578c25d7	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	3f6699c1-461e-483b-894d-7e04fa295a55	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	-	40e702d1-bea6-43de-aa3a-ba92664b26c8	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt

Click on the Key ID:


aws
Services ▼
AWSReservedSSO_AdministratorAccess_ed9c033dd25c6e5c/pebn@clou... ▼
Frankfurt ▼
Key Management Service (KMS)
AWS managed keys
Customer managed keys
Custom key stores
KMS > Customer managed keys > Key ID: 11181315-c749-4b59-abc6-bae8249b063e
Key actions ▼
Edit
Key policy
Tags
Key rotation
Edit

11181315-c749-4b59-abc6-bae8249b063e

General configuration

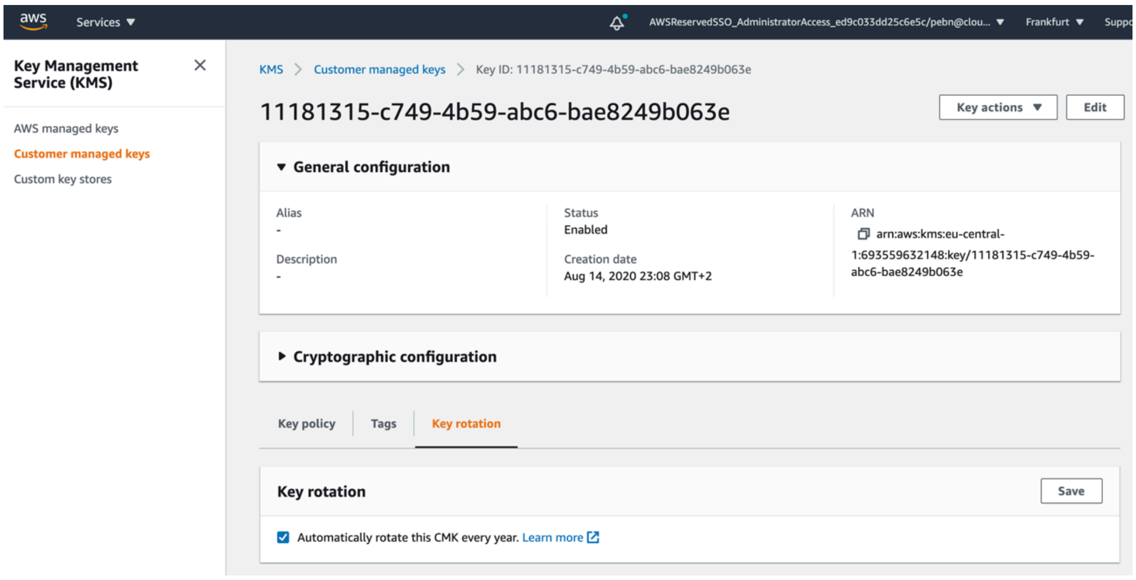
Alias: -
Description: -
Status: Enabled
Creation date: Aug 14, 2020 23:08 GMT+2
ARN: arn:aws:kms:eu-central-1:693559632148:key/11181315-c749-4b59-abc6-bae8249b063e

Cryptographic configuration

Key policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

Then click on the "Key rotation" tab:


aws
Services ▼
AWSReservedSSO_AdministratorAccess_ed9c033dd25c6e5c/pebn@clou... ▼
Frankfurt ▼
Key Management Service (KMS)
AWS managed keys
Customer managed keys
Custom key stores
KMS > Customer managed keys > Key ID: 11181315-c749-4b59-abc6-bae8249b063e
Key actions ▼
Edit
Key policy
Tags
Key rotation
Save

11181315-c749-4b59-abc6-bae8249b063e

General configuration

Alias: -
Description: -
Status: Enabled
Creation date: Aug 14, 2020 23:08 GMT+2
ARN: arn:aws:kms:eu-central-1:693559632148:key/11181315-c749-4b59-abc6-bae8249b063e

Cryptographic configuration

Key rotation

[✓] Automatically rotate this CMK every year. Learn more

Finally, make sure that the checkbox "Automatically rotate this CMK every year" is checked. If not, check it.

4 KMS Key Policies

KMS won't allow you to create a KMS Key with a policy that might lock you out from subsequently editing the key (quite apart from accessing it for encryption and decryption).

KMS will check when creating a key that the policy you have specified will (a) give you the required access you need for administration, and (b) that the Principal you specify exists. If not, deployment will fail.

A typical and perfectly valid policy statement for a KMS key would be:

```
{
  'Sid': 'Allow full KMS access for Admins',
  'Effect': 'Allow',
  'Principal': '*',
  'Action': ['kms:*'],
  'Resource': '*',
  'Condition':
    {
      'ArnLike':
        {
          'aws:PrincipalArn':
            {
              'Fn::Sub': 'arn:aws:iam::${AWS::AccountId}:role/aws-
reserved/sso.amazonaws.com/*/AWSReservedSSO_AdministratorAccess_*',
            },
        },
    },
}
```

However, the above statement is not enough. There are two reasons for this: (1) the AdministratorAccess role won't exist in all accounts, as SSO only sets up Roles where needed, and (2) it might take a few minutes before the Role becomes visible to KMS.

To get around this, we must use another statement like the following, which is the default policy statement created when setting these things up via the console. The idea is to grant the account root user access in case the other policies fail or an admin Role should be removed:

```
     {
       "Sid": "Enable IAM User Permissions",
       "Effect": "Allow",
       "Principal": { 'AWS': {'Fn::Sub':
                         'arn:aws:iam::${AWS::AccountId}:root'}},
       "Action": "kms:*",
       "Resource": "*"
     }
```

Adding this statement will allow the KMS policy to be created, even if there is no AdministratorAccess Role visible to KMS. However, the above policy might not be what you think it is.

4.1 Caveat

The above statement actually constitutes a security risk, as it doesn't just allow the root user full access to the KMS Key. It does that too - but what it really does is to enable access to KMS *for all users in the account*, not just to the root user.

It does not by itself give any IAM users or roles access to the CMK, but it enables admins to use IAM policies to do so. For more information, see https://docs.aws.amazon.com/kms/latest/developerguide/control-access-overview.html%23managing-access and also https://docs.aws.amazon.com/kms/latest/developerguide/key-policies.html%23key-policy-default.

You may find, if you use such a policy, that access is restricted to the account root user by means of a centrally installed SCP (Service Control Policy).

4.2 Sample KMS Policy

Below is a complete KMS Key policy intended to be used as a template. It has three policy statements, the first for the IAM Users in the account, the second one for KMS Key Administrators, and the third for granting access to a principal using the key for encryption and decryption.

You will want to modify the Principal of the last statement to fit your particular use case, as it currently gives access only to CloudTrail. The first statement should be kept as-is, but the second might need additional roles (Cf Appendix 1).

NB: It is extremely important that none of the three clauses provides too generous access. Most especially, do not try to cut corners by specifying “*” rights for all three types of KMS key users, as this would make users of the CMK key administrators. The range of Actions in each statement should give minimal permissions to (1) account users, (2) CMK key admins, and (3) CMK key users, respectively. Take care to preserve the distinctions.

```
        {
          'Version': '2012-10-17',
          'Id': 'accountcloudtrailkey',
          'Statement':
            [
              {
                'Sid': 'Allow full access for users in the same
account',
                'Effect': 'Allow',
                'Principal':
                  {
                    'AWS': { 'Fn::Sub':
'arn:aws:iam::${AWS::AccountId}:root' },
                  },
                'Action': 'kms:*',
                'Resource': '*',
              },

              {
                'Sid': 'Allow KMS key administrator access',
                'Effect': 'Allow',
                'Principal': '*',
                'Action':
                  [
                    'kms:Create*',
                    'kms:Describe*',
                    'kms:Enable*',
                    'kms:List*',
                    'kms:Put*',
                    'kms:Update*',
                    'kms:Revoke*',
                    'kms:Disable*',
                    'kms:Get*',
                    'kms:Delete*',
                    'kms:TagResource',
                    'kms:UntagResource',
                    'kms:ScheduleKeyDeletion',
                    'kms:CancelKeyDeletion',
                  ],
                'Resource': '*',
                'Condition':
                  {
                    'ArnLike':
                      {
                        'aws:PrincipalArn':
                          {
                            'Fn::Sub':
'arn:aws:iam::${AWS::AccountId}:role/aws-reserved/sso.amazonaws.com/*/AW
SReservedSSO_AdministratorAccess_*',
                          },
                      },
                  },
              },

              {
                'Sid': 'Allow use of the key',
                'Effect': 'Allow',
                'Principal': { 'Service': 'cloudtrail.amazonaws.com' },
                'Action':
                  [
                    'kms:Encrypt',
                    'kms:Decrypt',
                    'kms:ReEncrypt',
                    'kms:GenerateDataKey*',
                    'kms:DescribeKey',
                  ],
                'Resource': '*',
              },
            ],
        }
```

5 Other Considerations

5.1 Aliases

Consider using KMS aliases for your keys. They allow you to refer to your KMS keys using mnemonic names rather than an anonymous UUID.

For more information, see https://docs.aws.amazon.com/kms/latest/developerguide/kms-alias.html

5.2 Description

It is a good idea to always provide a description of the purpose of the KMS key. The examples in CloudFormation and Terraform in sections 3.1 and 3.2 give the syntax.

5.3 Do Not Delete Your KMS Keys Unduly

When you delete a KMS key, any data it has been used to encrypt cannot be decrypted again. In essence, access to all such data is lost.

For this reason, KMS keys are always removed with a time delay of a minimum of 7 days.

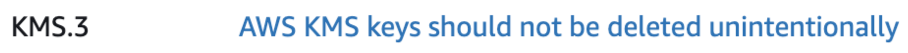
You can freely schedule KMS keys for deletion in the lower environments.

In the production environments, however, scheduling a KMS key for deletion creates a `CRITICAL` security incident in the form of a TEAMFIX ticket in Jira or ServiceNow. Basically, never delete a KMS key in production – we never want to lose production data because of a deleted key.

6 Security Hub Controls for KMS Keys

Below are the enabled controls in AWS Security Hub pertaining to KMS keys. Your keys must comply with all of them. Be proactive. It is easiest to make them compliant from the beginning, rather than when the automated security checks have created tickets for your team to fix misconfigurations which constitute security risks.

6.1 CRITICAL

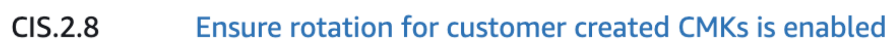


KMS.3	AWS KMS keys should not be deleted unintentionally

6.2 HIGH

–

6.3 MEDIUM



CIS.2.8	Ensure rotation for customer created CMKs is enabled

7 Appendix 1

7.1 The Problem

When using ordinary IAM Roles, typical architectural design in a multi-account environment is to let common Roles have the same name in every account. This allows us to write policy statements like the following:

```
      {
        'Sid': 'Allow full access for the FooBar IAM Role',
        'Effect': 'Allow',
        "Principal": { "AWS": "arn:aws:iam::123456789012:role/FooBar"
                     },
        'Action': ['some-service:*'],
        'Resource': '*'
      }
```

To make the above account-independent, modern practice is to use `!Sub` in YAML (or "`Fn::Sub`" in JSON) to substitute the real account number at policy creation time, like so:

```
      {
        'Sid': 'Allow full access for the FooBar IAM Role',
        'Effect': 'Allow',
        'Principal': { 'AWS': {'Fn::Sub':
             'arn:aws:iam::${AWS::AccountId}:role/FooBar' }},
        'Action': ['some-service:*'],
        'Resource': '*'
      }
```

(If you're still using `!Join` or "`Fn::Join`", switch to `!Sub`. There's no longer any need to suffer, and it's much more readable.)

However, under AWS SSO, a "Permission Set" is not a Role per se, it’s a template for a real Role in an account: the corresponding Role has a different name in every account. A Permission Set by the name of FooBar will have a Role created for it with an ARN similar to the following:

```
arn:aws:iam::123456789012:role/aws-reserved/sso.amazonaws.com/eu-central
-1/AWSReservedSSO_FooBar_f02384ab0f024b
```

The hex suffix will vary from account to account, so a first impulse would be the following:

```
{
  'Sid': 'Allow full access for the FooBar SSO Role',
  'Effect': 'Allow',
  'Principal': { 'AWS': {'Fn::Sub':

'arn:aws:iam::${AWS::AccountId}:role/aws-reserved/sso.amazonaws.com/*/AW
SReservedSSO_FooBar_*' }},
  'Action': ['some-service:*'],
  'Resource': '*'
}
```

The idea here is to use wildcards for the entropic hex noise and the region. Unfortunately, this won't work as wildcards are not permitted in Principal ARN strings.

7.2 The Solution

Instead, we must do the following:

```
{
  'Sid': 'Allow full access for the FooBar SSO Role',
  'Effect': 'Allow',
  'Principal': '*',
  'Action': ['some-service:*'],
  'Resource': '*',
  'Condition':
    {
      'ArnLike':
        {
          'aws:PrincipalArn':
            {
              'Fn::Sub': 'arn:aws:iam::${AWS::AccountId}:role/aws-
reserved/sso.amazonaws.com/*/AWSReservedSSO_FooBar_*',
            },
        },
    },
}
```

We grant access to all Principals, but then we restrict the allowed range of Principals using a Condition using ArnLike, which does accept wildcards. We need to specify the account id; using a wildcard there would give any AWS account with an SSO Admin access. However, we can safely wildcard the region.