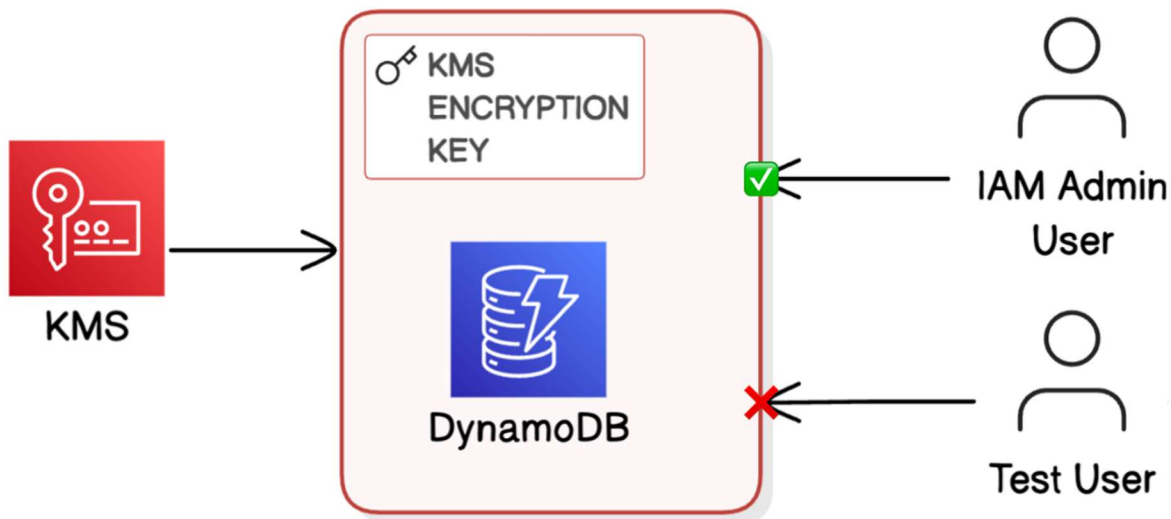


# Encrypt Data with AWS KMS

Protect a database using encryption with AWS KMS



## Introduction:

In this project, I will demonstrate using encryption to secure data. The goal is to create encryption keys with AWS KMS (Key Management Service), encrypt a DynamoDB tables data with that key, and test access using IAM Users.

## Tools and Concepts:

Services I used include AWS KMS (Key Management Service), DynamoDB, AWS IAM. Key concepts I learnt include encryption, database tables, KMS using permission to actions rather than just access to the key itself, creating a user to test access, encryption vs security groups vs permission policies and the way we can combine them together to create layers of security. This project took me approximately 1.5 hours including demo time. The most challenging part was understanding how encryption differs from other access control tools. It was most rewarding to see the test user get access again to the data tables.

I did this project today, to learn all about encryption, securing data, and how it actually works. This project gave me a foundation of encryption keys and managing access as an admin.

# Encryption and KMS:

Encryption is the process of turning original data into a secure format. Companies and developers do this to secure their data from unauthorized users. Encryption keys are the secure code that informs an algorithm on how it should encrypt.

AWS KMS is a vault for our encryption keys. Key Management Service like KMS are important because they help us secure and manage keys that we use to encrypt the data. Unauthorized access to the keys means exposing our encrypted data, which puts our security at risk.

Encryption keys are broadly categorized as symmetric and asymmetric. I set up symmetric keys because I will be using the exact same key to encrypt and decrypt our data. Whereas if we were to use the asymmetric key then that would use one key to encrypt and another key to decrypt it which is not required in this case as I am using the encryption to protect the data stored in a database, asymmetric is more useful when sharing data between two entities.

In this step, I will use AWS KMS (Key Management Service) to create an encryption key and make myself the owner/admin of that key.

Customer managed keys (1)						
<div>Filter keys by properties or tags</div>						
<input type="checkbox"/>	Aliases	Key ID	Status	Key type	Key spec	Key usage
<input type="checkbox"/>	<a href="#">atif-kms-key</a>	<a href="#">9ab15d1e-14a4-4866-a76d-b6d...</a>	Enabled	Symmetric	SYMMETRIC_DEFAULT	Encrypt and decrypt

# Encrypting Data:

Now, I will create a DynamoDB table, which will be the resource that I am securing using the encryption key I just generated.

My encryption key will safeguard data in DynamoDB, which is a fast and flexible AWS database service. DynamoDB is great for applications that need fast access to large amounts of data e.g. Gaming.

**Manage encryption** [Info](#)

All data stored in Amazon DynamoDB is fully encrypted at rest. By default, DynamoDB manages the encryption key, and you are not charged any fee for using it.

**Encryption at rest**

**Encryption key management**

☐

**AWS owned key**  
The key is owned and managed by DynamoDB. You are not charged additional fees for using this key.

☐

**AWS managed key**  
The key is stored in your account and managed by AWS Key Management Service (KMS). AWS KMS charges apply.

☒

**Customer managed key**  
The key is stored in your account and managed by you. AWS KMS charges apply.

atif-kms-key

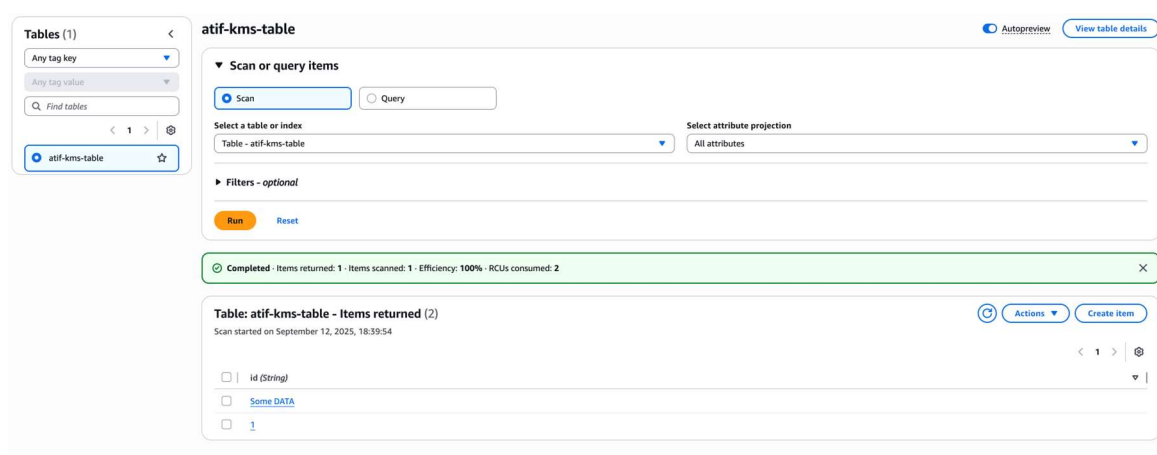
arn:aws:kms:us-east-2:789283941507:key/9ab15d1e-14a4-4866-a76d-b6d33294fad6

There are different encryption options in DynamoDB which include: DynamoDB managed, AWS managed, and Customer Managed Key (CMK). Their difference is based on who creates and manage the key; and I selected the CMK option so that I can manage it using the KMS key I created.

# Data Visibility:

Now, I will add data to the DynamoDB table and test the access to the encrypted data.

Rather than controlling who has access to the key KMS manages user permissions by controlling the actions the people can do with that key. In this case, even if I gave the user permission to see the key, they would still need permission to decrypt.



The screenshot displays the AWS IAM console interface for a scan operation on a DynamoDB table. On the left, a sidebar shows a list of tables, with 'atif-kms-table' selected. The main panel is titled 'atif-kms-table' and includes a 'Scan or query items' section. The 'Scan' option is selected, and the 'Table - atif-kms-table' is chosen for the scan. The 'Select attribute projection' is set to 'All attributes'. Below this, a 'Filters - optional' section is visible. A green status bar indicates the scan is 'Completed' with 1 item returned, 1 item scanned, 100% efficiency, and 2 RCUs consumed. The 'Table: atif-kms-table - Items returned (2)' section shows a table with one item: 'id (String)' with the value '1'. The scan started on September 12, 2025, at 18:39:54.

Despite encrypting my DynamoDB table I was able to still see the table's items because I am a User of the key. DynamoDB uses transparent data encryption, which means it does the encryption and decryption process for us because it knows that I am authorized to see the data.

# Testing:

## Denying Access:

Now, I will create a test User that does not have access to the encryption key i.e. does not have permission to encrypt/decrypt with the key. This will help to validate whether the encrypted data will still be visible to an unauthorized user.

**Users (2)** [Info](#)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

1 match

<input type="checkbox"/>	User name	Path	Group:	Last activity	MFA	Password age
<input type="checkbox"/>	<a href="#">test-kms-user</a>	/	0	-	-	-

I configured a new IAM user to validate whether unauthorized users can access encrypted data. The permission policies I granted this User are DynamoDB FULL access but not encryption/decryption permission with AWS KMS.

Next, I will try accessing the encrypted data as the test-kms-user (i.e. the unauthorized user). This is because I want to validate that the user won't get to see the data that should still be encrypted it.

**IAM user sign in** ⓘ

Account ID or alias [\(Don't have?\)](#)

☐ Remember this account

IAM username

Password

☐ Show Password [Having trouble?](#)

[Create a new AWS account](#)

Access denied to kms:Decrypt

You don't have permission to kms:Decrypt. To request access, copy the following text and send it to your AWS administrator. [Learn more about troubleshooting access denied errors.](#)

Diagnose with Amazon Q

User: arn:aws:iam::789283941507:user/test-kms-user

Action: kms:Decrypt

On resource(s): arn:aws:kms:us-east-2:789283941507:key/9ab15d1e-14a4-4866-a76d-b6d33294fad6

Context: no identity-based policy allows the action

After accessing the DynamoDB, I encountered an Access DENIED message, because my test user has no access to encrypted data (i.e. decryption). This proves that encryption keys can be used to secure data.

So, how does a KMS key Manage user permissions? -

Granting Access:

In this step, I will give the test user permission to encrypt using the encryption key. This will help me practice how I can use KMS to enable access (not just block people out).

Add key users

The following IAM users and roles can use this key to encrypt and decrypt data from within applications and when using AWS services integrated with KMS.

Key users (1/4)

test

1 matches

< 1 >

Name

Path

Type

test-kms-user

/

User

Cancel

Add

### Key policy

```
38     },
39     {
40       "Sid": "Allow use of the key",
41       "Effect": "Allow",
42       "Principal": {
43         "AWS": [
44           "arn:aws:iam::789283941507:user/Atif-IAM-Admin",
45           "arn:aws:iam::789283941507:user/test-kms-user"
46         ]
47       },
48       "Action": [
49         "kms:Encrypt",
50         "kms:Decrypt",
51         "kms:ReEncrypt*",
52         "kms:GenerateDataKey*",
53         "kms:DescribeKey"
54       ],
55       "Resource": "*"
56     },
57     {
58       "Sid": "Allow attachment of persistent resources",
59       "Effect": "Allow",
60       "Principal": {
61         "AWS": [
```

To let my test user use the encryption key, I made it KEY user in the KMS console. My key's policy was updated to allow the test-kms-user to encrypt, decrypt, re-encrypt, etc. using the key.

Then using the test user, I retried accessing the DynamoDB table and observed that the user is now able to see the data on the table. This confirms that making the user a KEY user is an effective way to authorize someone to access encrypted data.

### Table: atif-kms-table - Items returned (2)

Scan started on September 12, 2025, 21:15:33



Actions ▼

Create item

< 1 > ⚙

☐ | id (String)

☐ | [1](#)

☐ | [Some DATA](#)

# Encryption vs Other Access Controls:

Other access controls, like security groups or permission policies, control access to a resource (e.g. a DynamoDB table) or an entire service (e.g. DynamoDB).

However, these don't protect the actual data within the resource, like the items inside a DynamoDB table. Once someone has access, they can see all the data they're allowed to. These controls don't prevent someone with access from reading or misusing the data.

If we use KMS encryption, you secure the data within the resource. Even if someone bypasses the access controls (like hacking into a server or intercepting data), they can't understand the data unless they have the decryption key. This adds a layer of security - only users with access permissions and decryption keys can view the actual data.

In short Encryption secures data instead of an entire resource or service. We could combine encryption with other access control tool e.g. security groups and permission policies to have two layers of security – the resource level and the data level.

## Learning Outcomes:

- Create encryption keys with **AWS KMS (Key Management Service)**.
- ? Encrypt a **DynamoDB** database with a KMS key.
- Add and retrieve data from your database to **test** your encryption.
- Observe how AWS stops **unauthorized** access to your data.
- Give a user encryption access.

## Sources of Help:

Thank you [learn.nextwork.org](https://learn.nextwork.org) for helping with a step-by-step guide to complete this project.