Keys documentation

Azure Key Vault enables Microsoft Azure applications and users to store and use keys. It supports multiple key types and algorithms, and enables the use of Hardware Security Modules (HSM) for high value keys.

Keys

OVERVIEW

What are Key Vault keys?

Important news about soft-delete

Get started

QUICKSTART

Azure CLI

Azure PowerShell

Portal

.NET

Python

Go

Node.js

Java

Template

Tutorials

TUTORIAL

Configure key auto-rotation

Import HSM-protected keys (overview)

Client libraries for Key Vault keys

REFERENCE

.NET client library

Python client library

Java client library

Node.js client library

More about Key Vault

OVERVIEW

Key Vault overview

Basic concepts

Security features

Soft-delete overview

Microsoft Learn training

TRAINING

Manage secrets in your server apps with Azure Key Vault

About keys

Article • 01/25/2023 • 2 minutes to read

Azure Key Vault provides two types of resources to store and manage cryptographic keys. Vaults support software-protected and HSM-protected (Hardware Security Module) keys. Managed HSMs only support HSM-protected keys.

Resource type	Key protection methods	Data-plane endpoint base URL
Vaults	Software-protected and HSM-protected (with Premium SKU)	https://{vault-name}.vault.azure.net
Managed HSMs	HSM-protected	https://{hsm-name}.managedhsm.azure.net

- **Vaults** - Vaults provide a low-cost, easy to deploy, multi-tenant, zone-resilient (where available), highly available key management solution suitable for most common cloud application scenarios.
- **Managed HSMs** - Managed HSM provides single-tenant, zone-resilient (where available), highly available HSMs to store and manage your cryptographic keys. Most suitable for applications and usage scenarios that handle high value keys. Also helps to meet most stringent security, compliance, and regulatory requirements.

> **Note**
>
> Vaults also allow you to store and manage several types of objects like secrets, certificates and storage account keys, in addition to cryptographic keys.

Cryptographic keys in Key Vault are represented as JSON Web Key [JWK] objects. The JavaScript Object Notation (JSON) and JavaScript Object Signing and Encryption (JOSE) specifications are:

- JSON Web Key (JWK)
- JSON Web Encryption (JWE)
- JSON Web Algorithms (JWA)
- JSON Web Signature (JWS)

The base JWK/JWA specifications are also extended to enable key types unique to the Azure Key Vault and Managed HSM implementations.

HSM-protected keys (also referred to as HSM-keys) are processed in an HSM (Hardware Security Module) and always remain HSM protection boundary.

- Vaults use **FIPS 140-2 Level 2** validated HSMs to protect HSM-keys in shared HSM backend infrastructure.
- Managed HSM uses **FIPS 140-2 Level 3** validated HSM modules to protect your keys. Each HSM pool is an isolated single-tenant instance with its own security domain providing complete cryptographic isolation from all other HSMs sharing the same hardware infrastructure.

These keys are protected in single-tenant HSM-pools. You can import an RSA, EC, and symmetric key, in soft form or by exporting from a supported HSM device. You can also generate keys in HSM pools. When you import HSM keys using the method described in the BYOK (bring your own key) specification, it enables secure transportation key material into Managed HSM pools.

For more information on geographical boundaries, see Microsoft Azure Trust Center

Key types and protection methods

Key Vault supports RSA and EC keys. Managed HSM supports RSA, EC, and symmetric keys.

HSM-protected keys

Key type	Vaults (Premium SKU only)	Managed HSMs
EC-HSM: Elliptic Curve key	Supported (P-256, P-384, P-521, secp256k1/P-256K)	Supported (P-256, secp256k1/P-256K, P-384, P-521)
RSA-HSM: RSA key	Supported (2048-bit, 3072-bit, 4096-bit)	Supported (2048-bit, 3072-bit, 4096-bit)
oct-HSM: Symmetric key	Not supported	Supported (128-bit, 192-bit, 256-bit)

Software-protected keys

Key type	Vaults	Managed HSMs
RSA: "Software-protected" RSA key	Supported (2048-bit, 3072-bit, 4096-bit)	Not supported
EC: "Software-protected" Elliptic Curve key	Supported (P-256, P-384, P-521, secp256k1/P-256K)	Not supported

Compliance

Key type and destination	Compliance
Software-protected keys in vaults (Premium & Standard SKUs)	FIPS 140-2 Level 1
HSM-protected keys in vaults (Premium SKU)	FIPS 140-2 Level 2
HSM-protected keys in Managed HSM	FIPS 140-2 Level 3

See Key types, algorithms, and operations for details about each key type, algorithms, operations, attributes, and tags.

Usage Scenarios

When to use	Examples
Azure server-side data encryption for integrated resource providers with customer-managed keys	- Server-side encryption using customer-managed keys in Azure Key Vault
Client-side data encryption	- Client-Side Encryption with Azure Key Vault
Keyless TLS	- Use key Client Libraries

Next steps

- Key management in Azure
- About Key Vault
- About Managed HSM
- About secrets
- About certificates
- Key Vault REST API overview

- Authentication, requests, and responses
- Key Vault Developer's Guide

Additional resources

🕮 Documentation

Key types, algorithms, and operations - Azure Key Vault

Supported key types, algorithms, and operations (details).

Azure Key Vault Keys, Secrets, and Certificates Overview

Overview of Azure Key Vault REST interface and developer details for keys, secrets and certificates.

Best practices for secrets management - Azure Key Vault

Learn about best practices for Azure Key Vault secrets management.

Azure Managed HSM access control

Manage access permissions for Azure Managed HSM and keys. Covers the authentication and authorization model for Managed HSM, and how to secure your HSMs.

Azure Key Vault versions

The various versions of Azure Key Vault

Secure access to a managed HSM - Azure Key Vault Managed HSM

Learn how to secure access to Managed HSM using Azure RBAC and Managed HSM local RBAC

wrap Key - REST API (Azure Key Vault)

Learn more about [Key Vault wrap Key Operations]. How to [wrap Key].

Azure Quickstart - Set and retrieve a key from Key Vault using Azure portal

Quickstart showing how to set and retrieve a key from Azure Key Vault using the Azure portal

Show 5 more

Quickstart: Set and retrieve a key from Azure Key Vault using Azure CLI

Article • 01/13/2023 • 3 minutes to read

In this quickstart, you create a key vault in Azure Key Vault with Azure CLI. Azure Key Vault is a cloud service that works as a secure secrets store. You can securely store keys, passwords, certificates, and other secrets. For more information on Key Vault, review the Overview. Azure CLI is used to create and manage Azure resources using commands or scripts. Once that you've completed that, you will store a key.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This quickstart requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Use the az group create command to create a resource group named *myResourceGroup* in the *eastus* location.

Azure CLI

```
az group create --name "myResourceGroup" --location "EastUS"
```

Create a key vault

Use the Azure CLI az keyvault create command to create a Key Vault in the resource group from the previous step. You will need to provide some information:

- Key vault name: A string of 3 to 24 characters that can contain only numbers (0-9), letters (a-z, A-Z), and hyphens (-)

  > **ⓘ Important**
  >
  > Each key vault must have a unique name. Replace <your-unique-keyvault-name> with the name of your key vault in the following examples.

- Resource group name: **myResourceGroup**.
- The location: **EastUS**.

Azure CLI

```
az keyvault create --name "<your-unique-keyvault-name>" --resource-group
"myResourceGroup" --location "EastUS"
```

The output of this command shows properties of the newly created key vault. Take note of the two properties listed below:

- **Vault Name**: The name you provided to the --name parameter above.
- **Vault URI**: In the example, this is https://<your-unique-keyvault-name>.vault.azure.net/. Applications that use your vault through its REST API must use this URI.

At this point, your Azure account is the only one authorized to perform any operations on this new vault.

Add a key to Key Vault

To add a key to the vault, you just need to take a couple of additional steps. This key could be used by an application.

Type this command to create a key called **ExampleKey** :

Azure CLI

```
az keyvault key create --vault-name "<your-unique-keyvault-name>" -n
ExampleKey --protection software
```

You can now reference this key that you added to Azure Key Vault by using its URI. Use `https://<your-unique-keyvault-name>.vault.azure.net/keys/ExampleKey` to get the current version.

To view previously stored key:

Azure CLI

```
az keyvault key show --name "ExampleKey" --vault-name "<your-unique-
keyvault-name>"
```

Now, you've created a Key Vault, stored a key, and retrieved it.

Clean up resources

Other quickstarts and tutorials in this collection build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place.

When no longer needed, you can use the Azure CLI az group delete command to remove the resource group and all related resources:

Azure CLI

```
az group delete --name "myResourceGroup"
```

Next steps

In this quickstart, you created a Key Vault and stored a key in it. To learn more about Key Vault and how to integrate it with your applications, continue on to these articles.

- Read an Overview of Azure Key Vault
- See the reference for the Azure CLI az keyvault commands
- Review the Key Vault security overview

Additional resources

Documentation

Quickstart - Create an Azure Key Vault with the Azure CLI

Quickstart showing how to create an Azure Key Vault using the Azure CLI

Manage Azure Key Vault using CLI - Azure Key Vault

Use this article to automate common tasks in Key Vault by using the Azure CLI

Azure Quickstart - Set and retrieve a key from Key Vault using Azure portal

Quickstart showing how to set and retrieve a key from Azure Key Vault using the Azure portal

Quickstart - Set and retrieve a secret from Azure Key Vault

Quickstart showing how to set and retrieve a secret from Azure Key Vault using Azure CLI

az keyvault key

Tutorial: Use a managed identity to access Azure Key Vault - Linux - Microsoft Entra

A tutorial that walks you through the process of using a Linux VM system-assigned managed identity to access Azure Resource Manager.

REST API error codes - Azure Key Vault

These error codes could be returned by an operation on an Azure Key Vault web service.

Get Keys - REST API (Azure Key Vault)

Learn more about [Key Vault Get Keys Operations]. How to [Get Keys].

Show 5 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Certification

Microsoft Certified: Azure Administrator Associate - Certifications

Azure administrators implement, manage, and monitor an organization's Microsoft Azure

environment, including virtual networks, storage, compute, identity, security, and governance.

Quickstart: Set and retrieve a key from Azure Key Vault using Azure PowerShell

Article • 01/08/2023 • 3 minutes to read

In this quickstart, you create a key vault in Azure Key Vault with Azure PowerShell. Azure Key Vault is a cloud service that works as a secure secrets store. You can securely store keys, passwords, certificates, and other secrets. For more information on Key Vault, review the Overview. Azure PowerShell is used to create and manage Azure resources using commands or scripts. Once that you've completed that, you will store a key.

If you don't have an Azure subscription, create a free account before you begin.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.

4. Select **Enter** to run the code or command.

If you choose to install and use PowerShell locally, this tutorial requires Azure PowerShell module version 1.0.0 or later. Type `$PSVersionTable.PSVersion` to find the version. If you need to upgrade, see Install Azure PowerShell module. If you're running PowerShell locally, you also need to run `Login-AzAccount` to create a connection with Azure.

Azure PowerShell

```
Login-AzAccount
```

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. Use the Azure PowerShell New-AzResourceGroup cmdlet to create a resource group named *myResourceGroup* in the *eastus* location.

Azure PowerShell

```
New-AzResourceGroup -Name "myResourceGroup" -Location "EastUS"
```

Create a key vault

Use the Azure PowerShell New-AzKeyVault cmdlet to create a Key Vault in the resource group from the previous step. You will need to provide some information:

- Key vault name: A string of 3 to 24 characters that can contain only numbers (0-9), letters (a-z, A-Z), and hyphens (-)

  > **Important**
  >
  > Each key vault must have a unique name. Replace <your-unique-keyvault-name> with the name of your key vault in the following examples.

- Resource group name: **myResourceGroup**.
- The location: **EastUS**.

Azure PowerShell

```
New-AzKeyVault -Name "<your-unique-keyvault-name>" -ResourceGroupName
"myResourceGroup" -Location "EastUS"
```

The output of this cmdlet shows properties of the newly created key vault. Take note of the two properties listed below:

- **Vault Name**: The name you provided to the -Name parameter above.
- **Vault URI**: In the example, this is https://<your-unique-keyvault-name>.vault.azure.net/. Applications that use your vault through its REST API must use this URI.

At this point, your Azure account is the only one authorized to perform any operations on this new vault.

Add a key to Key Vault

To add a key to the vault, you just need to take a couple of additional steps. This key could be used by an application.

Type this command to create a called **ExampleKey** :

Azure PowerShell

```
Add-AzKeyVaultKey -VaultName "<your-unique-keyvault-name>" -Name
"ExampleKey" -Destination "Software"
```

You can now reference this key that you added to Azure Key Vault by using its URI. Use `https://<your-unique-keyvault-name>.vault.azure.net/keys/ExampleKey` to get the current version.

To view previously stored key:

Azure PowerShell

```
Get-AzKeyVaultKey -VaultName "<your-unique-keyvault-name>" -KeyName
"ExampleKey"
```

Now, you've created a Key Vault, stored a key, and retrieved it.

Clean up resources

Other quickstarts and tutorials in this collection build upon this quickstart. If you plan to continue on to work with other quickstarts and tutorials, you may want to leave these resources in place.

When no longer needed, you can use the Azure PowerShell Remove-AzResourceGroup cmdlet to remove the resource group and all related resources.

Azure PowerShell

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Next steps

In this quickstart, you created a Key Vault and stored a certificate in it. To learn more about Key Vault and how to integrate it with your applications, continue on to these articles.

- Read an Overview of Azure Key Vault
- See the reference for the Azure PowerShell Key Vault cmdlets
- Review the Key Vault security overview .md)

Additional resources

Documentation

Get-AzKeyVaultKey (Az.KeyVault)

The Get-AzKeyVaultKey cmdlet gets Azure Key Vault keys. This cmdlet gets a specific Microsoft.Azure.Commands.KeyVault.Models.KeyBundle or a list of all KeyBundle objects in a key vault or by version.

Get-AzKeyVault (Az.KeyVault)

The Get-AzKeyVault cmdlet gets information about the key vaults in a subscription. You can view all key vaults instances in a subscription, or filter your results by a resource group or a particular key vault. Note that although specifying the resource group is optional for this cmdlet when you get a...

Az.KeyVault Module

This topic displays help topics for the Azure Key Vault Cmdlets.

Set-AzKeyVaultAccessPolicy (Az.KeyVault)

The Set-AzKeyVaultAccessPolicy cmdlet grants or modifies existing permissions for a user, application, or security group to perform the specified operations with a key vault. It does not modify the permissions that other users, applications, or security groups have on the key vault. If you are...

Quickstart - Set & retrieve a secret from Key Vault using PowerShell

In this quickstart, learn how to create, retrieve, and delete secrets from an Azure Key Vault using Azure PowerShell.

Show 2 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Certification

Microsoft Certified: Azure Administrator Associate - Certifications

Azure administrators implement, manage, and monitor an organization's Microsoft Azure environment, including virtual networks, storage, compute, identity, security, and governance.

Quickstart: Set and retrieve a key from Azure Key Vault using the Azure portal

Article • 01/08/2023 • 2 minutes to read

Azure Key Vault is a cloud service that provides a secure store for secrets. You can securely store keys, passwords, certificates, and other secrets. Azure key vaults may be created and managed through the Azure portal. In this quickstart, you create a key vault, then use it to store a key. For more information on Key Vault, review the Overview.

Prerequisites

To access Azure Key Vault, you'll need an Azure subscription. If you don't already have a subscription, create a free account before you begin.

All access to secrets takes place through Azure Key Vault. For this quickstart, create a key vault using Azure portal, Azure CLI, or Azure PowerShell.

Sign in to Azure

Sign in to the Azure portal at https://portal.azure.com.

Add a key to Key Vault

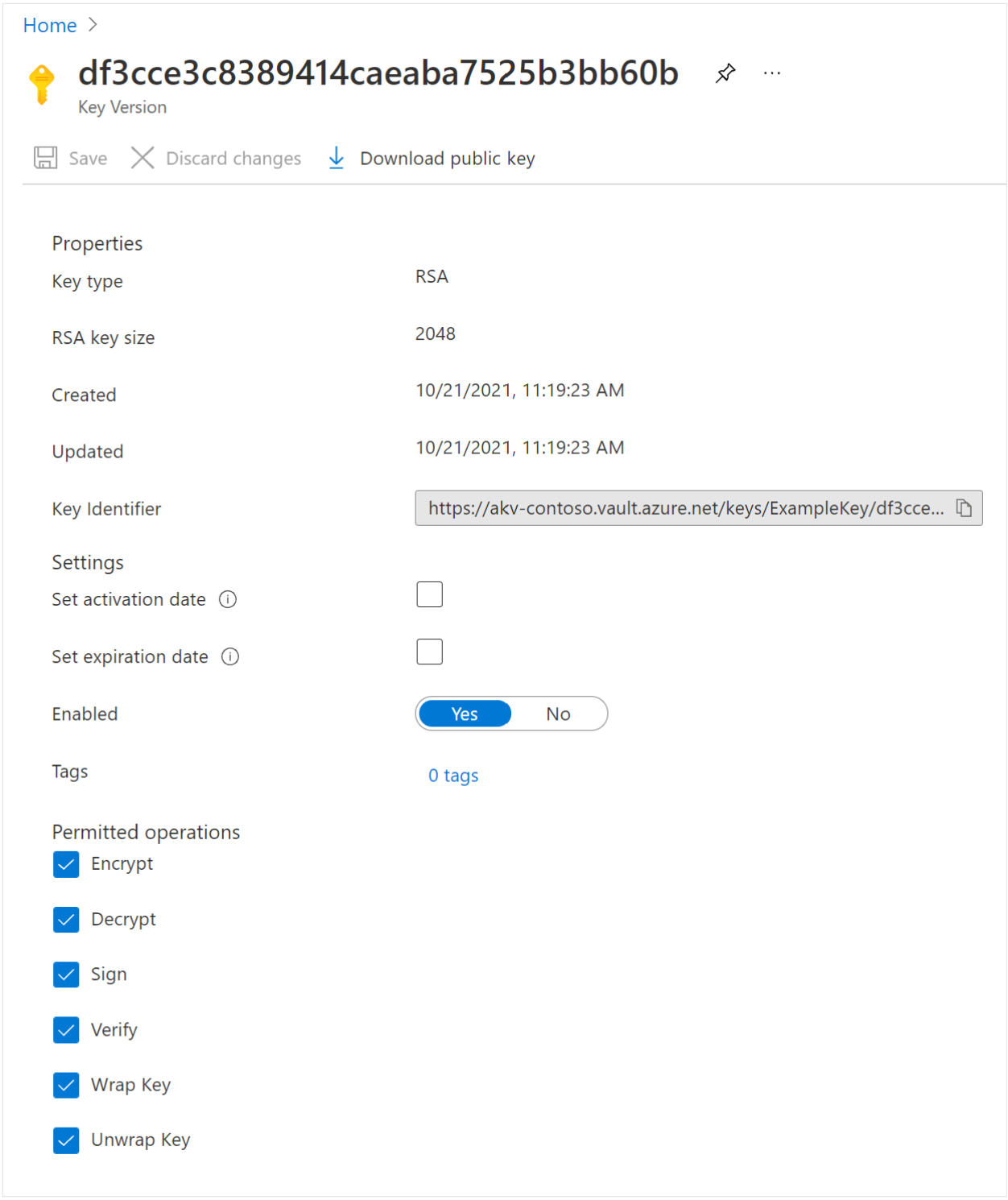
To add a key to the vault, you just need to take a couple of additional steps. In this case, we add a key that could be used by an application. The key is called **ExampleKey**.

1. On the Key Vault properties pages, select **Keys**.
2. Select **Generate/Import**.
3. On the **Create a key** screen choose the following values:

    - **Options**: Generate.
    - **Name**: ExampleKey.
    - Leave the other values to their defaults. Select **Create**.

Retrieve a key from Key Vault

Once that you receive the message that the key has been successfully created, you may click on it on the list. You can then see some of the properties and select **Download public key** to retrieve the key.



Clean up resources

Other Key Vault quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed, delete the resource group, which deletes the Key Vault and related resources. To delete the resource group through the portal:

1. Enter the name of your resource group in the Search box at the top of the portal. When you see the resource group used in this quickstart in the search results,

select it.

2. Select **Delete resource group**.
3. In the **TYPE THE RESOURCE GROUP NAME:** box type in the name of the resource group and select **Delete**.

Next steps

In this quickstart, you created a Key Vault and stored a key in it. To learn more about Key Vault and how to integrate it with your applications, continue on to these articles.

- Read an Overview of Azure Key Vault
- See the Azure Key Vault developer's guide
- Review the Key Vault security overview

Additional resources

🕮 Documentation

About Azure Key Vault secrets - Azure Key Vault

Overview of Azure Key Vault secrets.

Authenticate to Azure Key Vault

Learn how to authenticate to Azure Key Vault

REST API error codes - Azure Key Vault

These error codes could be returned by an operation on an Azure Key Vault web service.

Assign an Azure Key Vault access policy (CLI)

How to use the Azure CLI to assign a Key Vault access policy to a security principal or application identity.

Create and retrieve attributes of a key in Azure Key Vault - Azure CLI

Quickstart showing how to set and retrieve a key from Azure Key Vault using Azure CLI

Azure Key Vault Keys, Secrets, and Certificates Overview

Overview of Azure Key Vault REST interface and developer details for keys, secrets and certificates.

Azure Quickstart - Set and retrieve a secret from Key Vault using Azure portal

Quickstart showing how to set and retrieve a secret from Azure Key Vault using the Azure portal

Quickstart - Set and retrieve a secret from Azure Key Vault

Quickstart showing how to set and retrieve a secret from Azure Key Vault using Azure CLI

Show 5 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Quickstart: Azure Key Vault key client library for .NET

Article • 03/08/2023 • 5 minutes to read

Get started with the Azure Key Vault key client library for .NET. Azure Key Vault is a cloud service that provides a secure store for cryptographic keys. You can securely store cryptographic keys, passwords, certificates, and other secrets. Azure key vaults may be created and managed through the Azure portal. In this quickstart, you learn how to create, retrieve, and delete keys from an Azure key vault using the .NET key client library

Key Vault key client library resources:

API reference documentation | Library source code | Package (NuGet)

For more information about Key Vault and keys, see:

- Key Vault Overview
- Keys Overview.

Prerequisites

- An Azure subscription - create one for free
- .NET 6 SDK or later
- Azure CLI
- A Key Vault - you can create one using Azure portal, Azure CLI, or Azure PowerShell.

This quickstart is using `dotnet` and Azure CLI

Setup

This quickstart is using Azure Identity library with Azure CLI to authenticate user to Azure Services. Developers can also use Visual Studio or Visual Studio Code to authenticate their calls, for more information, see Authenticate the client with Azure Identity client library.

Sign in to Azure

1. Run the `login` command.

Azure CLI

```
az login
```

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at https://aka.ms/devicelogin and enter the authorization code displayed in your terminal.

2. Sign in with your account credentials in the browser.

Grant access to your key vault

Create an access policy for your key vault that grants key permissions to your user account

Azure CLI

```
az keyvault set-policy --name <your-key-vault-name> --upn user@domain.com --
key-permissions delete get list create purge
```

Create new .NET console app

1. In a command shell, run the following command to create a project named `key-vault-console-app`:

   .NET CLI

   ```
   dotnet new console --name key-vault-console-app
   ```

2. Change to the newly created *key-vault-console-app* directory, and run the following command to build the project:

   .NET CLI

   ```
   dotnet build
   ```

   The build output should contain no warnings or errors.

   Console

```
Build succeeded.
 0 Warning(s)
 0 Error(s)
```

Install the packages

From the command shell, install the Azure Key Vault key client library for .NET:

.NET CLI

```
dotnet add package Azure.Security.KeyVault.Keys
```

For this quickstart, you'll also need to install the Azure Identity client library:

.NET CLI

```
dotnet add package Azure.Identity
```

Set environment variables

This application is using key vault name as an environment variable called `KEY_VAULT_NAME`.

Windows

Windows Command Prompt

```
set KEY_VAULT_NAME=<your-key-vault-name>
```

Windows PowerShell

PowerShell

```
$Env:KEY_VAULT_NAME="<your-key-vault-name>"
```

macOS or Linux

Bash

```
export KEY_VAULT_NAME=<your-key-vault-name>
```

Object model

The Azure Key Vault key client library for .NET allows you to manage keys. The Code examples section shows how to create a client, set a key, retrieve a key, and delete a key.

Code examples

Add directives

Add the following directives to the top of *Program.cs*:

C#

```
using System;
using Azure.Identity;
using Azure.Security.KeyVault.Keys;
```

Authenticate and create a client

Application requests to most Azure services must be authorized. Using the DefaultAzureCredential class provided by the Azure Identity client library is the recommended approach for implementing passwordless connections to Azure services in your code. `DefaultAzureCredential` supports multiple authentication methods and determines which method should be used at runtime. This approach enables your app to use different authentication methods in different environments (local vs. production) without implementing environment-specific code.

In this quickstart, `DefaultAzureCredential` authenticates to key vault using the credentials of the local development user logged into the Azure CLI. When the application is deployed to Azure, the same `DefaultAzureCredential` code can automatically discover and use a managed identity that is assigned to an App Service, Virtual Machine, or other services. For more information, see Managed Identity Overview.

In this example, the name of your key vault is expanded to the key vault URI, in the format `https://<your-key-vault-name>.vault.azure.net`. For more information about authenticating to key vault, see Developer's Guide.

C#

```
var keyVaultName = Environment.GetEnvironmentVariable("KEY_VAULT_NAME");
var kvUri = $"https://{keyVaultName}.vault.azure.net";

var client = new KeyClient(new Uri(kvUri), new DefaultAzureCredential());
```

Save a key

For this task, use the CreateKeyAsync method. The method's parameters accepts a key name and the key type.

C#

```
var key = await client.CreateKeyAsync("myKey", KeyType.Rsa);
```

> ⓘ Note
>
> If key name exists, this code will create new version of that key.

Retrieve a key

You can now retrieve the previously created key with the GetKeyAsync method.

C#

```
var key = await client.GetKeyAsync("myKey");
```

Delete a key

Finally, let's delete and purge the key from your key vault with the StartDeleteKeyAsync and PurgeDeletedKeyAsync methods.

C#

```
var operation = await client.StartDeleteKeyAsync("myKey");

// You only need to wait for completion if you want to purge or recover the
key.
await operation.WaitForCompletionAsync();

var key = operation.Value;
await client.PurgeDeletedKeyAsync("myKey");
```

Sample code

Modify the .NET console app to interact with the Key Vault by completing the following steps:

- Replace the code in *Program.cs* with the following code:

C#

```
using System;
using System.Threading.Tasks;
using Azure.Identity;
using Azure.Security.KeyVault.Keys;

namespace key_vault_console_app
{
    class Program
    {
        static async Task Main(string[] args)
        {
            const string keyName = "myKey";
            var keyVaultName =
Environment.GetEnvironmentVariable("KEY_VAULT_NAME");
            var kvUri = $"https://{keyVaultName}.vault.azure.net";

            var client = new KeyClient(new Uri(kvUri), new
DefaultAzureCredential());

            Console.Write($"Creating a key in {keyVaultName} called
'{keyName}' ...");
            var createdKey = await client.CreateKeyAsync(keyName,
KeyType.Rsa);
            Console.WriteLine("done.");

            Console.WriteLine($"Retrieving your key from
{keyVaultName}.");
            var key = await client.GetKeyAsync(keyName);
            Console.WriteLine($"Your key version is
'{key.Value.Properties.Version}'.");

            Console.Write($"Deleting your key from {keyVaultName}
...");
            var deleteOperation = await
client.StartDeleteKeyAsync(keyName);
            // You only need to wait for completion if you want to
purge or recover the key.
            await deleteOperation.WaitForCompletionAsync();
            Console.WriteLine("done.");

            Console.Write($"Purging your key from {keyVaultName} ...");
            await client.PurgeDeletedKeyAsync(keyName);
            Console.WriteLine(" done.");
        }
```

```
    }
}
```

Test and verify

1. Execute the following command to build the project

   .NET CLI

   ```
   dotnet build
   ```

2. Execute the following command to run the app.

   .NET CLI

   ```
   dotnet run
   ```

3. When prompted, enter a secret value. For example, mySecretPassword.

   A variation of the following output appears:

   Console

   ```
   Creating a key in mykeyvault called 'myKey' ... done.
   Retrieving your key from mykeyvault.
   Your key version is '8532359bced24e4bb2525f2d2050738a'.
   Deleting your key from jl-kv ... done
   Purging your key from <your-unique-keyvault-name> ... done.
   ```

Next steps

In this quickstart, you created a key vault, stored a key, and retrieved that key.

To learn more about Key Vault and how to integrate it with your apps, see the following articles:

- Read an Overview of Azure Key Vault
- Read an Overview of keys
- See an Access Key Vault from App Service Application Tutorial
- See an Access Key Vault from Virtual Machine Tutorial
- See the Azure Key Vault developer's guide
- Review the Key Vault security overview

Additional resources

Documentation

Azure.Security.KeyVault.Secrets samples for .NET - Code Samples

Samples for the Azure.Security.KeyVault.Secrets client library.

Azure Key Vault SDK for .NET - Azure for .NET Developers

Reference for Azure Key Vault SDK for .NET

Azure Key Vault secret client library for .NET - Azure for .NET Developers

Azure Key Vault key client library for .NET - Azure for .NET Developers

Quickstart - Azure Key Vault secrets client library for .NET

Learn how to create, retrieve, and delete secrets from an Azure key vault using the .NET client library

Azure.Security.KeyVault.Secrets Namespace - Azure for .NET Developers

Explore all classes and interfaces of the Azure.Security.KeyVault.Secrets namespace.

KeyVaultClient Class (Microsoft.Azure.KeyVault) - Azure for .NET Developers

Client class to perform cryptographic key operations and vault operations against the Key Vault service.

Azure.Security.KeyVault.Keys samples for .NET - Code Samples

Samples for the Azure.Security.KeyVault.Keys client library.

Show 5 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Certification

Microsoft Certified: Azure Developer Associate - Certifications

Azure developers design, build, test, and maintain cloud applications and services.

Quickstart: Azure Key Vault key client library for JavaScript

Article • 02/03/2023 • 5 minutes to read

Get started with the Azure Key Vault key client library for JavaScript. Azure Key Vault is a cloud service that provides a secure store for cryptographic keys. You can securely store keys, passwords, certificates, and other secrets. Azure key vaults may be created and managed through the Azure portal. In this quickstart, you learn how to create, retrieve, and delete keys from an Azure key vault using the JavaScript key client library

Key Vault client library resources:

API reference documentation | Library source code | Package (npm)

For more information about Key Vault and keys, see:

- Key Vault Overview
- Keys Overview.

Prerequisites

- An Azure subscription - create one for free.
- Current Node.js LTS.
- Azure CLI
- An existing Key Vault - you can create one using:
  - Azure CLI
  - Azure portal
  - Azure PowerShell

This quickstart assumes you're running Azure CLI.

Sign in to Azure

1. Run the `login` command.

```
Azure CLI
```

```
az login
```

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at https://aka.ms/devicelogin and enter the authorization code displayed in your terminal.

2. Sign in with your account credentials in the browser.

Create new Node.js application

Create a Node.js application that uses your key vault.

1. In a terminal, create a folder named `key-vault-node-app` and change into that folder:

```
terminal
mkdir key-vault-node-app && cd key-vault-node-app
```

2. Initialize the Node.js project:

```
terminal
npm init -y
```

Install Key Vault packages

1. Using the terminal, install the Azure Key Vault secrets client library, @azure/keyvault-keys for Node.js.

```
terminal
npm install @azure/keyvault-keys
```

2. Install the Azure Identity client library, @azure/identity package to authenticate to a Key Vault.

```
terminal
npm install @azure/identity
```

Grant access to your key vault

Create an access policy for your key vault that grants key permissions to your user account

Azure CLI

```
az keyvault set-policy --name <YourKeyVaultName> --upn user@domain.com --
key-permissions delete get list create update purge
```

Set environment variables

This application is using key vault name as an environment variable called `KEY_VAULT_NAME`.

Windows

Windows Command Prompt

```
set KEY_VAULT_NAME=<your-key-vault-name>
```

Authenticate and create a client

Application requests to most Azure services must be authorized. Using the DefaultAzureCredential method provided by the Azure Identity client library is the recommended approach for implementing passwordless connections to Azure services in your code. `DefaultAzureCredential` supports multiple authentication methods and determines which method should be used at runtime. This approach enables your app to use different authentication methods in different environments (local vs. production) without implementing environment-specific code.

In this quickstart, `DefaultAzureCredential` authenticates to key vault using the credentials of the local development user logged into the Azure CLI. When the application is deployed to Azure, the same `DefaultAzureCredential` code can automatically discover and use a managed identity that is assigned to an App Service, Virtual Machine, or other services. For more information, see Managed Identity Overview.

In this code, the name of your key vault is used to create the key vault URI, in the format `https://<your-key-vault-name>.vault.azure.net`. For more information about authenticating to key vault, see Developer's Guide.

Code example

The code samples below will show you how to create a client, set a secret, retrieve a secret, and delete a secret.

This code uses the following Key Vault Secret classes and methods:

- DefaultAzureCredential class
- KeyClient class
  - createKey
  - createEcKey
  - createRsaKey
  - getKey
  - listPropertiesOfKeys
  - updateKeyProperties
  - beginDeleteKey
  - getDeletedKey
  - purgeDeletedKey

Set up the app framework

1. Create new text file and paste the following code into the **index.js** file.

JavaScript

```
const { KeyClient } = require("@azure/keyvault-keys");
const { DefaultAzureCredential } = require("@azure/identity");

async function main() {

    // DefaultAzureCredential expects the following three environment
variables:
    // - AZURE_TENANT_ID: The tenant ID in Azure Active Directory
    // - AZURE_CLIENT_ID: The application (client) ID registered in the
AAD tenant
    // - AZURE_CLIENT_SECRET: The client secret for the registered
application
    const credential = new DefaultAzureCredential();

    const keyVaultName = process.env["KEY_VAULT_NAME"];
    if(!keyVaultName) throw new Error("KEY_VAULT_NAME is empty");
    const url = "https://" + keyVaultName + ".vault.azure.net";
```

```
    const client = new KeyClient(url, credential);

    const uniqueString = Date.now();
    const keyName = `sample-key-${uniqueString}`;
    const ecKeyName = `sample-ec-key-${uniqueString}`;
    const rsaKeyName = `sample-rsa-key-${uniqueString}`;

    // Create key using the general method
    const result = await client.createKey(keyName, "EC");
    console.log("key: ", result);

    // Create key using specialized key creation methods
    const ecResult = await client.createEcKey(ecKeyName, { curve: "P-
256" });
    const rsaResult = await client.createRsaKey(rsaKeyName, { keySize:
2048 });
    console.log("Elliptic curve key: ", ecResult);
    console.log("RSA Key: ", rsaResult);

    // Get a specific key
    const key = await client.getKey(keyName);
    console.log("key: ", key);

    // Or list the keys we have
    for await (const keyProperties of client.listPropertiesOfKeys()) {
    const key = await client.getKey(keyProperties.name);
    console.log("key: ", key);
    }

    // Update the key
    const updatedKey = await client.updateKeyProperties(keyName,
result.properties.version, {
    enabled: false
    });
    console.log("updated key: ", updatedKey);

    // Delete the key - the key is soft-deleted but not yet purged
    const deletePoller = await client.beginDeleteKey(keyName);
    await deletePoller.pollUntilDone();

    const deletedKey = await client.getDeletedKey(keyName);
    console.log("deleted key: ", deletedKey);

    // Purge the key - the key is permanently deleted
    // This operation could take some time to complete
    console.time("purge a single key");
    await client.purgeDeletedKey(keyName);
    console.timeEnd("purge a single key");
}

main().catch((error) => {
  console.error("An error occurred:", error);
  process.exit(1);
});
```

Run the sample application

1. Run the app:

```
terminal
```

```
node index.js
```

2. The create and get methods return a full JSON object for the key:

```
JSON
```

```
"key":  {
  "key": {
    "kid": "https://YOUR-KEY-VAULT-NAME.vault.azure.net/keys/YOUR-KEY-
NAME/YOUR-KEY-VERSION",
    "kty": "YOUR-KEY-TYPE",
    "keyOps": [ ARRAY-OF-VALID-OPERATIONS ],
    ... other properties based on key type
  },
  "id": "https://YOUR-KEY-VAULT-NAME.vault.azure.net/keys/YOUR-KEY-
NAME/YOUR-KEY-VERSION",
  "name": "YOUR-KEY-NAME",
  "keyOperations": [ ARRAY-OF-VALID-OPERATIONS ],
  "keyType": "YOUR-KEY-TYPE",
  "properties": {
    "tags": undefined,
    "enabled": true,
    "notBefore": undefined,
    "expiresOn": undefined,
    "createdOn": 2021-11-29T18:29:11.000Z,
    "updatedOn": 2021-11-29T18:29:11.000Z,
    "recoverableDays": 90,
    "recoveryLevel": "Recoverable+Purgeable",
    "exportable": undefined,
    "releasePolicy": undefined,
    "vaultUrl": "https://YOUR-KEY-VAULT-NAME.vault.azure.net",
    "version": "YOUR-KEY-VERSION",
    "name": "YOUR-KEY-VAULT-NAME",
    "managed": undefined,
    "id": "https://YOUR-KEY-VAULT-NAME.vault.azure.net/keys/YOUR-KEY-
NAME/YOUR-KEY-VERSION"
  }
}
```

Integrating with App Configuration

The Azure SDK provides a helper method, parseKeyVaultKeyIdentifier, to parse the given Key Vault Key ID. This is necessary if you use App Configuration references to Key Vault. App Config stores the Key Vault Key ID. You need the *parseKeyVaultKeyIdentifier* method to parse that ID to get the key name. Once you have the key name, you can get the current key value using code from this quickstart.

Next steps

In this quickstart, you created a key vault, stored a key, and retrieved that key. To learn more about Key Vault and how to integrate it with your applications, continue on to these articles.

- Read an Overview of Azure Key Vault
- Read an Overview of Azure Key Vault Keys
- How to Secure access to a key vault
- See the Azure Key Vault developer's guide
- Review the Key Vault security overview

Additional resources

Documentation

Azure Key Vault Key client library for JavaScript

Azure Key Vault Secret client library for JavaScript

How to set and get secrets from Azure Key Vault using Node.js - Code Samples

How to set and get secrets from Azure Key Vault using Node.js.

SecretClient class

The SecretClient provides methods to manage KeyVaultSecret in the Azure Key Vault. The client supports creating, retrieving, updating, deleting, purging, backing up, restoring and listing KeyVaultSecrets. The client also supports listing DeletedSecret for a soft-delete enabled Azure Key...

Quickstart - Azure Key Vault secret client library for JavaScript (version 4)

Learn how to create, retrieve, and delete secrets from an Azure key vault using the JavaScript client library

@azure/keyvault-secrets package

Azure Key Vault Keys client library samples for TypeScript - Code Samples

These sample programs show how to use the TypeScript client libraries for Azure Key Vault Keys in

some common scenarios.

Show 5 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Quickstart: Azure Key Vault keys client library for Python

Article • 03/15/2023 • 5 minutes to read

Get started with the Azure Key Vault client library for Python. Follow these steps to install the package and try out example code for basic tasks. By using Key Vault to store cryptographic keys, you avoid storing such keys in your code, which increases the security of your app.

API reference documentation | Library source code | Package (Python Package Index)

Prerequisites

- An Azure subscription - create one for free.
- Python 3.7+
- Azure CLI

This quickstart assumes you're running Azure CLI or Azure PowerShell in a Linux terminal window.

Set up your local environment

This quickstart is using the Azure Identity library with Azure CLI or Azure PowerShell to authenticate the user to Azure services. Developers can also use Visual Studio or Visual Studio Code to authenticate their calls. For more information, see Authenticate the client with Azure Identity client library.

Sign in to Azure

Azure CLI

1. Run the `login` command.

   Azure CLI

   ```
   az login
   ```

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at https://aka.ms/devicelogin and enter the authorization code displayed in your terminal.

2. Sign in with your account credentials in the browser.

Install the packages

1. In a terminal or command prompt, create a suitable project folder, and then create and activate a Python virtual environment as described on Use Python virtual environments.

2. Install the Azure Active Directory identity library:

```
terminal

pip install azure-identity
```

3. Install the Key Vault key client library:

```
terminal

pip install azure-keyvault-keys
```

Create a resource group and key vault

Azure CLI

1. Use the `az group create` command to create a resource group:

```
Azure CLI

az group create --name myResourceGroup --location eastus
```

You can change "eastus" to a location nearer to you, if you prefer.

2. Use `az keyvault create` to create the key vault:

```
Azure CLI
```

```
az keyvault create --name <your-unique-keyvault-name> --resource-
group myResourceGroup
```

Replace `<your-unique-keyvault-name>` with a name that's unique across all of Azure. You typically use your personal or company name along with other numbers and identifiers.

Set the KEY_VAULT_NAME environmental variable

Our script will use the value assigned to the `KEY_VAULT_NAME` environment variable as the name of the key vault. You must therefore set this value using the following command:

Console

```
export KEY_VAULT_NAME=<your-unique-keyvault-name>
```

Grant access to your key vault

Create an access policy for your key vault that grants key permission to your user account.

Azure CLI

Azure CLI

```
az keyvault set-policy --name <your-unique-keyvault-name> --upn
user@domain.com --key-permissions get list create delete
```

Create the sample code

The Azure Key Vault key client library for Python allows you to manage cryptographic keys. The following code sample demonstrates how to create a client, set a key, retrieve a key, and delete a key.

Create a file named *kv_keys.py* that contains this code.

Python

```
import os
from azure.keyvault.keys import KeyClient
from azure.identity import DefaultAzureCredential

keyVaultName = os.environ["KEY_VAULT_NAME"]
KVUri = "https://" + keyVaultName + ".vault.azure.net"

credential = DefaultAzureCredential()
client = KeyClient(vault_url=KVUri, credential=credential)

keyName = input("Input a name for your key > ")

print(f"Creating a key in {keyVaultName} called '{keyName}' ...")

rsa_key = client.create_rsa_key(keyName, size=2048)

print(" done.")

print(f"Retrieving your key from {keyVaultName}.")

retrieved_key = client.get_key(keyName)

print(f"Key with name '{retrieved_key.name}' was found.")
print(f"Deleting your key from {keyVaultName} ...")

poller = client.begin_delete_key(keyName)
deleted_key = poller.result()

print(" done.")
```

Run the code

Make sure the code in the previous section is in a file named *kv_keys.py*. Then run the code with the following command:

terminal

```
python kv_keys.py
```

- If you encounter permissions errors, make sure you ran the az keyvault set-policy or Set-AzKeyVaultAccessPolicy command.
- Rerunning the code with the same key name may produce the error, "(Conflict) Key <name> is currently in a deleted but recoverable state." Use a different key name.

Code details

Authenticate and create a client

Application requests to most Azure services must be authorized. Using the DefaultAzureCredential class provided by the Azure Identity client library is the recommended approach for implementing passwordless connections to Azure services in your code. `DefaultAzureCredential` supports multiple authentication methods and determines which method should be used at runtime. This approach enables your app to use different authentication methods in different environments (local vs. production) without implementing environment-specific code.

In this quickstart, `DefaultAzureCredential` authenticates to key vault using the credentials of the local development user logged into the Azure CLI. When the application is deployed to Azure, the same `DefaultAzureCredential` code can automatically discover and use a managed identity that is assigned to an App Service, Virtual Machine, or other services. For more information, see Managed Identity Overview.

In the example code, the name of your key vault is expanded using the value of the `KVUri` variable, in the format: "https://<your-key-vault-name>.vault.azure.net".

Python

```
credential = DefaultAzureCredential()
client = KeyClient(vault_url=KVUri, credential=credential)
```

Save a key

Once you've obtained the client object for the key vault, you can store a key using the create_rsa_key method:

Python

```
rsa_key = client.create_rsa_key(keyName, size=2048)
```

You can also use create_key or create_ec_key.

Calling a `create` method generates a call to the Azure REST API for the key vault.

When Azure handles the request, it authenticates the caller's identity (the service principal) using the credential object you provided to the client.

Retrieve a key

To read a key from Key Vault, use the get_key method:

Python

```
retrieved_key = client.get_key(keyName)
```

You can also verify that the key has been set with the Azure CLI command az keyvault key show or the Azure PowerShell cmdlet Get-AzKeyVaultKey.

Delete a key

To delete a key, use the begin_delete_key method:

Python

```
poller = client.begin_delete_key(keyName)
deleted_key = poller.result()
```

The `begin_delete_key` method is asynchronous and returns a poller object. Calling the poller's `result` method waits for its completion.

You can verify that the key is deleted with the Azure CLI command az keyvault key show or the Azure PowerShell cmdlet Get-AzKeyVaultKey.

Once deleted, a key remains in a deleted but recoverable state for a time. If you run the code again, use a different key name.

Clean up resources

If you want to also experiment with certificates and secrets, you can reuse the Key Vault created in this article.

Otherwise, when you're finished with the resources created in this article, use the following command to delete the resource group and all its contained resources:

Azure CLI

Azure CLI

```
az group delete --resource-group myResourceGroup
```

Next steps

- Overview of Azure Key Vault
- Secure access to a key vault
- Azure Key Vault developer's guide
- Key Vault security overview
- Authenticate with Key Vault

Additional resources

Documentation

Azure Key Vault Keys client library for Python

Quickstart – Azure Key Vault Python client library – manage secrets

Learn how to create, retrieve, and delete secrets from an Azure key vault using the Python client library

Azure Key Vault SDK for Python

Reference for Azure Key Vault SDK for Python

Azure Key Vault Secrets client library for Python

How to set and get secrets from Azure Key Vault with Azure Managed Identities and Python - Code Samples

How to set and get secrets from Azure Key Vault with Azure Managed Identities and Python.

Azure Key Vault Keys Client Library Python Samples - Code Samples

azure.mgmt.keyvault package

Quickstart – Azure Key Vault Python client library – manage certificates

Learn how to create, retrieve, and delete certificates from an Azure key vault using the Python client library

Show 5 more

Training

Module
Implement Azure Key Vault - Training

Implement Azure Key Vault

Certification

Microsoft Certified: Azure Developer Associate - Certifications

Azure developers design, build, test, and maintain cloud applications and services.

Quickstart: Azure Key Vault keys client library for Go

Article • 07/13/2022 • 3 minutes to read

In this quickstart, you'll learn to use the Azure SDK for Go to create, retrieve, update, list, and delete Azure Key Vault keys.

Azure Key Vault is a cloud service that works as a secure secrets store. You can securely store keys, passwords, certificates, and other secrets. For more information on Key Vault, you may review the Overview.

Follow this guide to learn how to use the azkeys package to manage your Azure Key Vault keys using Go.

Prerequisites

- An Azure subscription - create one for free.
- **Go installed**: Version 1.18 or above
- Azure CLI

Set up your environment

1. Sign into Azure.

   Azure CLI

   ```
   az login
   ```

2. Create a new resource group.

   Azure CLI

   ```
   az group create --name quickstart-rg --location eastus
   ```

3. Deploy a new key vault instance.

   Azure CLI

   ```
   az keyvault create --name <keyVaultName> --resource-group quickstart-rg
   ```

Replace `<keyVaultName>` with a name that's unique across all of Azure. You typically use your personal or company name along with other numbers and identifiers.

4. Create a new Go module and install packages

Azure CLI

```
go mod init quickstart-keys
go get -u github.com/Azure/azure-sdk-for-go/sdk/azidentity
go get -u github.com/Azure/azure-sdk-for-go/sdk/keyvault/azkeys
```

Create the sample code

Create a file named main.go and copy the following code into the file:

Go

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"

    "github.com/Azure/azure-sdk-for-go/sdk/azcore/to"
    "github.com/Azure/azure-sdk-for-go/sdk/azidentity"
    "github.com/Azure/azure-sdk-for-go/sdk/keyvault/azkeys"
)

func main() {
    keyVaultName := os.Getenv("KEY_VAULT_NAME")
    keyVaultUrl := fmt.Sprintf("https://%s.vault.azure.net/", keyVaultName)

    // create credential
    cred, err := azidentity.NewDefaultAzureCredential(nil)
    if err != nil {
        log.Fatalf("failed to obtain a credential: %v", err)
    }

    // create azkeys client
    client := azkeys.NewClient(keyVaultUrl, cred, nil)

    // create RSA Key
    rsaKeyParams := azkeys.CreateKeyParameters{
        Kty:     to.Ptr(azkeys.JSONWebKeyTypeRSA),
        KeySize: to.Ptr(int32(2048)),
    }
    rsaResp, err := client.CreateKey(context.TODO(), "new-rsa-key",
rsaKeyParams, nil)
```

```
    if err != nil {
        log.Fatalf("failed to create rsa key: %v", err)
    }
    fmt.Printf("New RSA key ID: %s\n", *rsaResp.Key.KID)

    // create EC Key
    ecKeyParams := azkeys.CreateKeyParameters{
        Kty:   to.Ptr(azkeys.JSONWebKeyTypeEC),
        Curve: to.Ptr(azkeys.JSONWebKeyCurveNameP256),
    }
    ecResp, err := client.CreateKey(context.TODO(), "new-ec-key",
ecKeyParams, nil)
    if err != nil {
        log.Fatalf("failed to create ec key: %v", err)
    }
    fmt.Printf("New EC key ID: %s\n", *ecResp.Key.KID)

    // list all vault keys
    fmt.Println("List all vault keys:")
    pager := client.NewListKeysPager(nil)
    for pager.More() {
        page, err := pager.NextPage(context.TODO())
        if err != nil {
            log.Fatal(err)
        }
        for _, key := range page.Value {
            fmt.Println(*key.KID)
        }
    }

    // update key properties to disable key
    updateParams := azkeys.UpdateKeyParameters{
        KeyAttributes: &azkeys.KeyAttributes{
            Enabled: to.Ptr(false),
        },
    }
    // an empty string version updates the latest version of the key
    version := ""
    updateResp, err := client.UpdateKey(context.TODO(), "new-rsa-key",
version, updateParams, nil)
    if err != nil {
        panic(err)
    }
    fmt.Printf("Key %s Enabled attribute set to: %t\n", *updateResp.Key.KID,
*updateResp.Attributes.Enabled)

    // delete the created keys
    for _, keyName := range []string{"new-rsa-key", "new-ec-key"} {
        // DeleteKey returns when Key Vault has begun deleting the key. That
can take several
        // seconds to complete, so it may be necessary to wait before
performing other operations
        // on the deleted key.
        delResp, err := client.DeleteKey(context.TODO(), keyName, nil)
        if err != nil {
```

```
            panic(err)
        }
        fmt.Printf("Successfully deleted key %s", *delResp.Key.KID)
    }
}
```

Run the code

Before you run the code, create an environment variable named KEY_VAULT_NAME. Set the environment variable's value to the name of the Azure Key Vault created previously.

Bash

```
export KEY_VAULT_NAME=quickstart-kv
```

Next, run the following `go run` command to run the app:

Bash

```
go run main.go
```

Output

```
Key ID: https://quickstart-kv.vault.azure.net/keys/new-rsa-
key4/f78fe1f34b064934bac86cc8c66a75c3: Key Type: RSA
Key ID: https://quickstart-kv.vault.azure.net/keys/new-ec-
key2/10e2cec51d1749c0a26aab784808cfaf: Key Type: EC
List all vault keys:
https://quickstart-kv.vault.azure.net/keys/new-ec-key
https://quickstart-kv.vault.azure.net/keys/new-ec-key1
https://quickstart-kv.vault.azure.net/keys/new-ec-key2
https://quickstart-kv.vault.azure.net/keys/new-rsa-key4
Enabled set to: false
Successfully deleted key https://quickstart-kv.vault.azure.net/keys/new-rsa-
key4/f78fe1f34b064934bac86cc8c66a75c3
```

Note

The output is for informational purposes only. Your return values may vary based on your Azure subscription and Azure Key Vault.

Code examples

See the module documentation for more examples.

Clean up resources

Run the following command to delete the resource group and all its remaining resources:

Azure CLI

```
az group delete --resource-group quickstart-rg
```

Next steps

- Overview of Azure Key Vault
- Secure access to a key vault
- Azure Key Vault developer's guide
- Key Vault security overview
- Authenticate with Key Vault

Additional resources

Documentation

Quickstart: Manage secrets by using the Azure Key Vault Go client library

Learn how to create, retrieve, and delete secrets from an Azure key vault by using the Go client library.

Create and retrieve attributes of a key in Azure Key Vault - Azure CLI

Quickstart showing how to set and retrieve a key from Azure Key Vault using Azure CLI

Azure Key Vault Certificate client library for Java

az keyvault key

Quickstart - Create an Azure Key Vault with the Azure CLI

Quickstart showing how to create an Azure Key Vault using the Azure CLI

Quickstart - Azure Key Vault Key client library for Java

Provides a quickstart for the Azure Key Vault Keys client library for Java.

Tutorial: Use a managed identity to access Azure Key Vault - Linux - Microsoft Entra

A tutorial that walks you through the process of using a Linux VM system-assigned managed identity

to access Azure Resource Manager.

Use Azure Key Vault to deliver TLS/SSL certificates to the JVM

Use Azure Key Vault to deliver TLS/SSL certificates to the JVM

Show 5 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Certification

Microsoft Certified: Azure Developer Associate - Certifications

Azure developers design, build, test, and maintain cloud applications and services.

Quickstart: Azure Key Vault Key client library for Java

Article • 01/23/2023 • 5 minutes to read

Get started with the Azure Key Vault Key client library for Java. Follow these steps to install the package and try out example code for basic tasks.

Additional resources:

- Source code
- API reference documentation
- Product documentation
- Samples

Prerequisites

- An Azure subscription - create one for free .
- Java Development Kit (JDK) version 8 or above
- Apache Maven
- Azure CLI

This quickstart assumes you're running Azure CLI and Apache Maven in a Linux terminal window.

Setting up

This quickstart is using the Azure Identity library with Azure CLI to authenticate user to Azure Services. Developers can also use Visual Studio or Visual Studio Code to authenticate their calls, for more information, see Authenticate the client with Azure Identity client library.

Sign in to Azure

1. Run the `login` command.

Azure CLI

```
az login
```

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at https://aka.ms/devicelogin and enter the authorization code displayed in your terminal.

2. Sign in with your account credentials in the browser.

Create a new Java console app

In a console window, use the `mvn` command to create a new Java console app with the name `akv-keys-java`.

Console

```
mvn archetype:generate -DgroupId=com.keyvault.keys.quickstart
                       -DartifactId=akv-keys-java
                       -DarchetypeArtifactId=maven-archetype-quickstart
                       -DarchetypeVersion=1.4
                       -DinteractiveMode=false
```

The output from generating the project will look something like this:

Console

```
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype:
maven-archetype-quickstart:1.4
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.keyvault.keys.quickstart
[INFO] Parameter: artifactId, Value: akv-keys-java
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.keyvault.keys.quickstart
[INFO] Parameter: packageInPathFormat, Value: com/keyvault/quickstart
[INFO] Parameter: package, Value: com.keyvault.keys.quickstart
[INFO] Parameter: groupId, Value: com.keyvault.keys.quickstart
[INFO] Parameter: artifactId, Value: akv-keys-java
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: /home/user/quickstarts/akv-
keys-java
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  38.124 s
[INFO] Finished at: 2019-11-15T13:19:06-08:00
```

```
[INFO] ------------------------------------------------------------------------
---
```

Change your directory to the newly created `akv-keys-java/` folder.

Console

```
cd akv-keys-java
```

Install the package

Open the *pom.xml* file in your text editor. Add the following dependency elements to the group of dependencies.

XML

```
    <dependency>
      <groupId>com.azure</groupId>
      <artifactId>azure-security-keyvault-keys</artifactId>
      <version>4.2.3</version>
    </dependency>

    <dependency>
      <groupId>com.azure</groupId>
      <artifactId>azure-identity</artifactId>
      <version>1.2.0</version>
    </dependency>
```

Create a resource group and key vault

This quickstart uses a pre-created Azure key vault. You can create a key vault by following the steps in the Azure CLI quickstart, Azure PowerShell quickstart, or Azure portal quickstart.

Alternatively, you can simply run the Azure CLI or Azure PowerShell commands below.

> **ⓘ Important**
>
> Each key vault must have a unique name. Replace <your-unique-keyvault-name> with the name of your key vault in the following examples.

Azure CLI

Azure CLI

```
az group create --name "myResourceGroup" -l "EastUS"

az keyvault create --name "<your-unique-keyvault-name>" -g
"myResourceGroup"
```

Grant access to your key vault

Create an access policy for your key vault that grants key permissions to your user account.

Azure CLI

```
az keyvault set-policy --name <your-key-vault-name> --upn user@domain.com --
key-permissions delete get list create purge
```

Set environment variables

This application is using your key vault name as an environment variable called `KEY_VAULT_NAME`.

Windows

Windows Command Prompt

```
set KEY_VAULT_NAME=<your-key-vault-name>
```

Windows PowerShell

PowerShell

```
$Env:KEY_VAULT_NAME="<your-key-vault-name>"
```

macOS or Linux

Windows Command Prompt

```
export KEY_VAULT_NAME=<your-key-vault-name>
```

Object model

The Azure Key Vault Key client library for Java allows you to manage keys. The Code examples section shows how to create a client, create a key, retrieve a key, and delete a key.

The entire console app is supplied in Sample code.

Code examples

Add directives

Add the following directives to the top of your code:

Java

```
import com.azure.core.util.polling.SyncPoller;
import com.azure.identity.DefaultAzureCredentialBuilder;

import com.azure.security.keyvault.keys.KeyClient;
import com.azure.security.keyvault.keys.KeyClientBuilder;
import com.azure.security.keyvault.keys.models.DeletedKey;
import com.azure.security.keyvault.keys.models.KeyType;
import com.azure.security.keyvault.keys.models.KeyVaultKey;
```

Authenticate and create a client

Application requests to most Azure services must be authorized. Using the DefaultAzureCredential class is the recommended approach for implementing passwordless connections to Azure services in your code. `DefaultAzureCredential` supports multiple authentication methods and determines which method should be used at runtime. This approach enables your app to use different authentication methods in different environments (local vs. production) without implementing environment-specific code.

In this quickstart, `DefaultAzureCredential` authenticates to key vault using the credentials of the local development user logged into the Azure CLI. When the application is deployed to Azure, the same `DefaultAzureCredential` code can automatically discover and use a managed identity that is assigned to an App Service, Virtual Machine, or other services. For more information, see Managed Identity Overview.

In this example, the name of your key vault is expanded to the key vault URI, in the format `https://<your-key-vault-name>.vault.azure.net`. For more information about

authenticating to key vault, see Developer's Guide.

Java

```
String keyVaultName = System.getenv("KEY_VAULT_NAME");
String keyVaultUri = "https://" + keyVaultName + ".vault.azure.net";

KeyClient keyClient = new KeyClientBuilder()
    .vaultUrl(keyVaultUri)
    .credential(new DefaultAzureCredentialBuilder().build())
    .buildClient();
```

Create a key

Now that your application is authenticated, you can create a key in your key vault using the `keyClient.createKey` method. This requires a name for the key and a key type. We've assigned the value "myKey" to the `keyName` variable and use a an RSA `KeyType` in this sample.

Java

```
keyClient.createKey(keyName, KeyType.RSA);
```

You can verify that the key has been set with the az keyvault key show command:

Azure CLI

```
az keyvault key show --vault-name <your-unique-key-vault-name> --name myKey
```

Retrieve a key

You can now retrieve the previously created key with the `keyClient.getKey` method.

Java

```
KeyVaultKey retrievedKey = keyClient.getKey(keyName);
```

You can now access the details of the retrieved key with operations like `retrievedKey.getProperties`, `retrievedKey.getKeyOperations`, etc.

Delete a key

Finally, let's delete the key from your key vault with the `keyClient.beginDeleteKey` method.

Key deletion is a long running operation, for which you can poll its progress or wait for it to complete.

Java

```
SyncPoller<DeletedKey, Void> deletionPoller =
keyClient.beginDeleteKey(keyName);
deletionPoller.waitForCompletion();
```

You can verify that the key has been deleted with the az keyvault key show command:

Azure CLI

```
az keyvault key show --vault-name <your-unique-key-vault-name> --name myKey
```

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove your key vault and the corresponding resource group.

Azure CLI

```
az group delete -g "myResourceGroup"
```

Azure PowerShell

```
Remove-AzResourceGroup -Name "myResourceGroup"
```

Sample code

Java

```
package com.keyvault.keys.quickstart;

import com.azure.core.util.polling.SyncPoller;
import com.azure.identity.DefaultAzureCredentialBuilder;

import com.azure.security.keyvault.keys.KeyClient;
import com.azure.security.keyvault.keys.KeyClientBuilder;
import com.azure.security.keyvault.keys.models.DeletedKey;
```

```
import com.azure.security.keyvault.keys.models.KeyType;
import com.azure.security.keyvault.keys.models.KeyVaultKey;

public class App {
    public static void main(String[] args) throws InterruptedException,
IllegalArgumentException {
        String keyVaultName = System.getenv("KEY_VAULT_NAME");
        String keyVaultUri = "https://" + keyVaultName + ".vault.azure.net";

        System.out.printf("key vault name = %s and key vault URI = %s \n",
keyVaultName, keyVaultUri);

        KeyClient keyClient = new KeyClientBuilder()
             .vaultUrl(keyVaultUri)
             .credential(new DefaultAzureCredentialBuilder().build())
             .buildClient();

        String keyName = "myKey";

        System.out.print("Creating a key in " + keyVaultName + " called '" +
keyName + " ... ");

        keyClient.createKey(keyName, KeyType.RSA);

        System.out.print("done.");
        System.out.println("Retrieving key from " + keyVaultName + ".");

        KeyVaultKey retrievedKey = keyClient.getKey(keyName);

        System.out.println("Your key's ID is '" + retrievedKey.getId() +
"'.");
        System.out.println("Deleting your key from " + keyVaultName + " ...
");

        SyncPoller<DeletedKey, Void> deletionPoller =
keyClient.beginDeleteKey(keyName);
        deletionPoller.waitForCompletion();

        System.out.print("done.");
    }
}
```

Next steps

In this quickstart, you created a key vault, created a key, retrieved it, and then deleted it. To learn more about Key Vault and how to integrate it with your applications, continue on to these articles.

- Read an Overview of Azure Key Vault
- Read the Key Vault security overview

- See the Azure Key Vault developer's guide
- How to Secure access to a key vault

Additional resources

Documentation

Quickstart - Azure Key Vault Secret client library for Java

Provides a quickstart for the Azure Key Vault Secret client library for Java.

Azure Key Vault Certificate client library for Java

Azure Key Vault Secret client library for Java

Azure Key Vault Key client library for Java

Azure Key Vault SDK for Java

Reference for Azure Key Vault SDK for Java

Azure Key Vault Secrets configuration properties

This reference doc contains all Azure Key Vault Secrets configuration properties.

Azure Key Vault Secret Samples client library for Java - Code Samples

This document explains samples and how to use them.

Quickstart for Azure Key Vault Certificate client library - Java

Learn about the the Azure Key Vault Certificate client library for Java with the steps in this quickstart.

Show 5 more

Training

Module

Implement Azure Key Vault - Training

Implement Azure Key Vault

Certification

Microsoft Certified: Azure Developer Associate - Certifications

Azure developers design, build, test, and maintain cloud applications and services.

Quickstart: Create an Azure key vault and a key by using Bicep

Article • 04/13/2023

Azure Key Vault is a cloud service that provides a secure store for secrets, such as keys, passwords, and certificate. This quickstart focuses on the process of deploying a Bicep file to create a key vault and a key.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

To complete this article:

- If you don't have an Azure subscription, create a free account before you begin.
- User would need to have an Azure built-in role assigned, recommended role **contributor**. Learn more here

Review the Bicep file

Bicep

```
@description('The name of the key vault to be created.')
param vaultName string

@description('The name of the key to be created.')
param keyName string

@description('The location of the resources')
param location string = resourceGroup().location

@description('The SKU of the vault to be created.')
@allowed([
  'standard'
  'premium'
])
param skuName string = 'standard'

@description('The JsonWebKeyType of the key to be created.')
@allowed([
  'EC'
```

```
  'EC-HSM'
  'RSA'
  'RSA-HSM'
])
param keyType string = 'RSA'

@description('The permitted JSON web key operations of the key to be
created.')
param keyOps array = []

@description('The size in bits of the key to be created.')
param keySize int = 2048

@description('The JsonWebKeyCurveName of the key to be created.')
@allowed([
  ''
  'P-256'
  'P-256K'
  'P-384'
  'P-521'
])
param curveName string = ''

resource vault 'Microsoft.KeyVault/vaults@2021-11-01-preview' = {
  name: vaultName
  location: location
  properties: {
    accessPolicies:[]
    enableRbacAuthorization: false
    enableSoftDelete: false
    enabledForDeployment: false
    enabledForDiskEncryption: false
    enabledForTemplateDeployment: false
    tenantId: subscription().tenantId
    sku: {
      name: skuName
      family: 'A'
    }
    networkAcls: {
      defaultAction: 'Allow'
      bypass: 'AzureServices'
    }
  }
}

resource key 'Microsoft.KeyVault/vaults/keys@2021-11-01-preview' = {
  parent: vault
  name: keyName
  properties: {
    kty: keyType
    keyOps: keyOps
    keySize: keySize
    curveName: curveName
  }
}
```

```
output proxyKey object = key.properties
```

Two resources are defined in the Bicep file:

- Microsoft.KeyVault/vaults
- Microsoft.KeyVault/vaults/keys

More Azure Key Vault template samples can be found in Azure Quickstart Templates.

Parameters and definitions

Parameter	Definition
keyOps	Specifies operations that can be performed by using the key. If you don't specify this parameter, all operations can be performed. The acceptable values for this parameter are a comma-separated list of key operations as defined by the JSON Web Key (JWK) specification: `["sign", "verify", "encrypt", "decrypt", " wrapKey", "unwrapKey"]`
CurveName	Elliptic curve (EC) name for EC key type. See JsonWebKeyCurveName
Kty	The type of key to create. For valid values, see JsonWebKeyType
Tags	Application-specific metadata in the form of key-value pairs.
nbf	Specifies the time, as a DateTime object, before which the key can't be used. The format would be Unix time stamp (the number of seconds after Unix Epoch on January 1st, 1970 at UTC).
exp	Specifies the expiration time, as a DateTime object. The format would be Unix time stamp (the number of seconds after Unix Epoch on January 1st, 1970 at UTC).

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

CLI

Azure CLI

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-
```

```
file main.bicep --parameters vaultName=<vault-name> keyName=<key-
name>
```

> **Note**
>
> Replace **<vault-name>** with the name of the key vault. Replace **<vault-name>** with the name of the key vault, and replace **<key-name>** with the name of the key.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

You can use the Azure portal to check the key vault and the key. Alternatively, use the following Azure CLI or Azure PowerShell script to list the key created.

CLI

Azure CLI

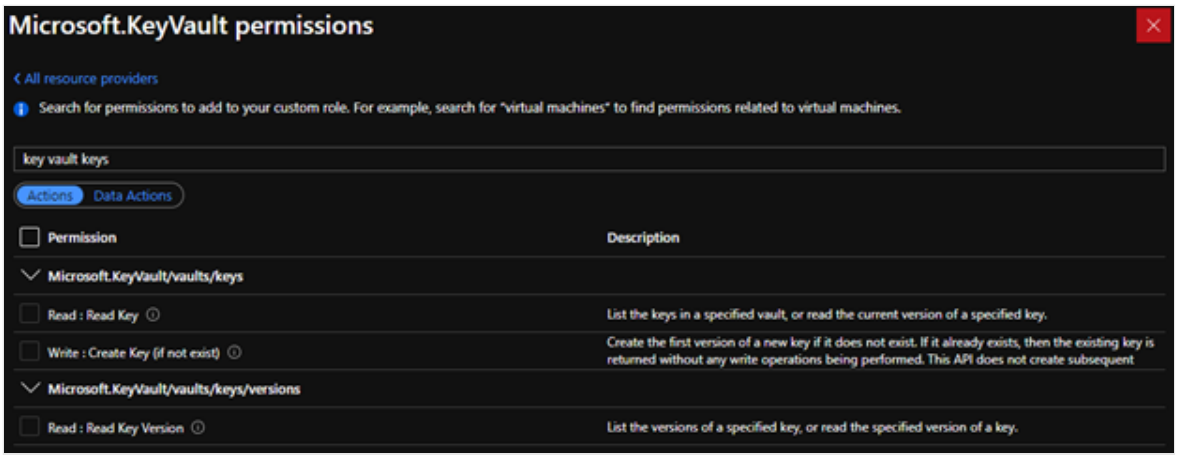
```
echo "Enter your key vault name:" &&
read keyVaultName &&
az keyvault key list --vault-name $keyVaultName &&
echo "Press [ENTER] to continue ..."
```

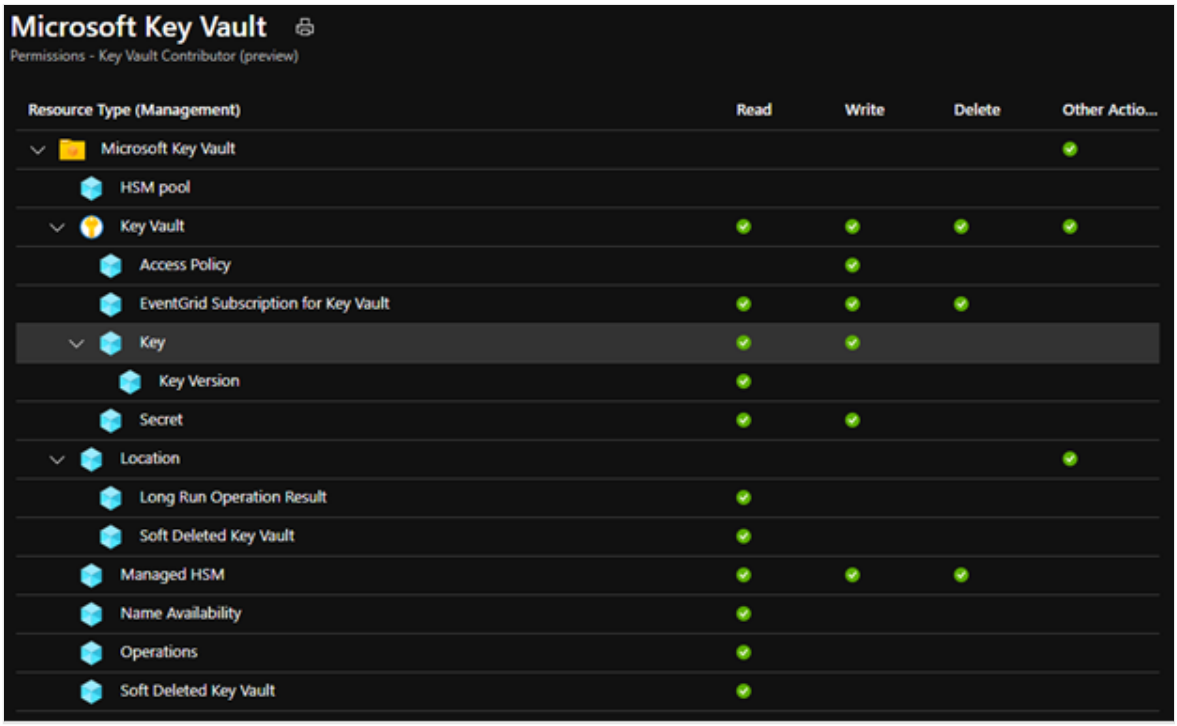
Creating key using ARM template is different from creating key via data plane

Creating a key via ARM

- It's only possible to create *new* keys. It isn't possible to update existing keys, nor create new versions of existing keys. If the key already exists, then the existing key is retrieved from storage and used (no write operations will occur).
- To be authorized to use this API, the caller needs to have the "**Microsoft.KeyVault/vaults/keys/write**" role-based access control (RBAC) Action.

The built-in "Key Vault Contributor" role is sufficient, since it authorizes all RBAC Actions that match the pattern "Microsoft.KeyVault/*".





Existing API (creating key via data plane)

- It's possible to create new keys, update existing keys, and create new versions of existing keys.
- The caller must be authorized to use this API. If the vault uses access policies, the caller must have "create" key permission; if the vault is enabled for RBAC, the caller must have "Microsoft.KeyVault/vaults/keys/create/action" RBAC DataAction.

Clean up resources

Other Key Vault quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed, delete the resource group, which

deletes the Key Vault and related resources. To delete the resource group by using Azure CLI or Azure PowerShell:

CLI

Azure CLI

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName &&
echo "Press [ENTER] to continue ..."
```

Next steps

In this quickstart, you created a key vault and a key using a Bicep file, and validated the deployment. To learn more about Key Vault and Azure Resource Manager, see these articles.

- Read an Overview of Azure Key Vault
- Learn more about Azure Resource Manager
- Review the Key Vault security overview

Quickstart: Create an Azure key vault and a key by using ARM template

Article • 03/08/2023 • 4 minutes to read

Azure Key Vault is a cloud service that provides a secure store for secrets, such as keys, passwords, and certificate. This quickstart focuses on the process of deploying an Azure Resource Manager template (ARM template) to create a key vault and a key.

Prerequisites

To complete this article:

- If you don't have an Azure subscription, create a free account before you begin.
- User would need to have an Azure built-in role assigned, recommended role **contributor**. Learn more here

Review the template

JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "vaultName": {
      "type": "string",
      "metadata": {
        "description": "The name of the key vault to be created."
      }
    },
    "keyName": {
      "type": "string",
      "metadata": {
        "description": "The name of the key to be created."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "The location of the resources"
      }
    },
    "skuName": {
```

```
      "type": "string",
      "defaultValue": "standard",
      "allowedValues": [
        "standard",
        "premium"
      ],
      "metadata": {
        "description": "The SKU of the vault to be created."
      }
    },
    "keyType": {
      "type": "string",
      "defaultValue": "RSA",
      "allowedValues": [
        "EC",
        "EC-HSM",
        "RSA",
        "RSA-HSM"
      ],
      "metadata": {
        "description": "The JsonWebKeyType of the key to be created."
      }
    },
    "keyOps": {
      "type": "array",
      "defaultValue": [],
      "metadata": {
        "description": "The permitted JSON web key operations of the key to
be created."
      }
    },
    "keySize": {
      "type": "int",
      "defaultValue": 2048,
      "metadata": {
        "description": "The size in bits of the key to be created."
      }
    },
    "curveName": {
      "type": "string",
      "defaultValue": "",
      "allowedValues": [
        "",
        "P-256",
        "P-256K",
        "P-384",
        "P-521"
      ],
      "metadata": {
        "description": "The JsonWebKeyCurveName of the key to be created."
      }
    }
  },
  "resources": [
    {
```

```
      "type": "Microsoft.KeyVault/vaults",
      "apiVersion": "2021-11-01-preview",
      "name": "[parameters('vaultName')]",
      "location": "[parameters('location')]",
      "properties": {
        "accessPolicies": [],
        "enableRbacAuthorization": false,
        "enableSoftDelete": false,
        "enabledForDeployment": false,
        "enabledForDiskEncryption": false,
        "enabledForTemplateDeployment": false,
        "tenantId": "[subscription().tenantId]",
        "sku": {
          "name": "[parameters('skuName')]",
          "family": "A"
        },
        "networkAcls": {
          "defaultAction": "Allow",
          "bypass": "AzureServices"
        }
      }
    },
    {
      "type": "Microsoft.KeyVault/vaults/keys",
      "apiVersion": "2021-11-01-preview",
      "name": "[format('{0}/{1}', parameters('vaultName'),
parameters('keyName'))]",
      "properties": {
        "kty": "[parameters('keyType')]",
        "keyOps": "[parameters('keyOps')]",
        "keySize": "[parameters('keySize')]",
        "curveName": "[parameters('curveName')]"
      },
      "dependsOn": [
        "[resourceId('Microsoft.KeyVault/vaults', parameters('vaultName'))]"
      ]
    }
  ],
  "outputs": {
    "proxyKey": {
      "type": "object",
      "value": "[reference(resourceId('Microsoft.KeyVault/vaults/keys',
parameters('vaultName'), parameters('keyName')))]"
    }
  }
}
```

Two resources are defined in the template:

- Microsoft.KeyVault/vaults
- Microsoft.KeyVault/vaults/keys

More Azure Key Vault template samples can be found in Azure Quickstart Templates.

Parameters and definitions

Parameter	Definition
keyOps	Specifies operations that can be performed by using the key. If you don't specify this parameter, all operations can be performed. The acceptable values for this parameter are a comma-separated list of key operations as defined by the JSON Web Key (JWK) specification: `["sign", "verify", "encrypt", "decrypt", " wrapKey", "unwrapKey"]`
CurveName	Elliptic curve (EC) name for EC key type. See JsonWebKeyCurveName
Kty	The type of key to create. For valid values, see JsonWebKeyType
Tags	Application-specific metadata in the form of key-value pairs.
nbf	Specifies the time, as a DateTime object, before which the key can't be used. The format would be Unix time stamp (the number of seconds after Unix Epoch on January 1st, 1970 at UTC).
exp	Specifies the expiration time, as a DateTime object. The format would be Unix time stamp (the number of seconds after Unix Epoch on January 1st, 1970 at UTC).

Deploy the template

You can use Azure portal, Azure PowerShell, Azure CLI, or REST API. To learn about deployment methods, see Deploy templates.

Review deployed resources

You can use the Azure portal to check the key vault and the key. Alternatively, use the following Azure CLI or Azure PowerShell script to list the key created.

CLI

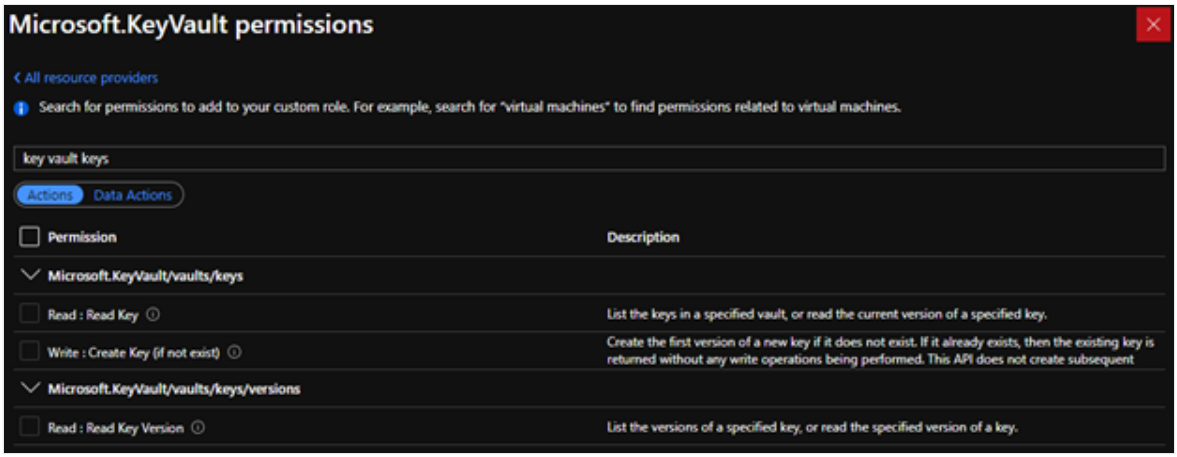
Azure CLI

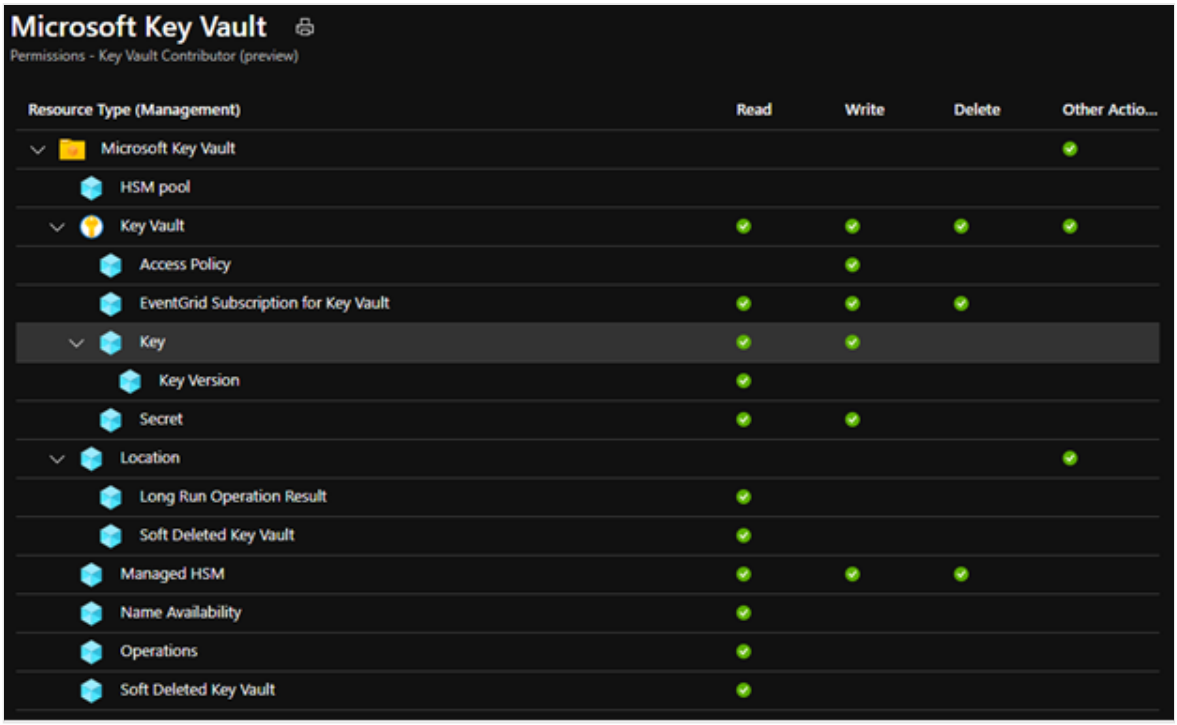
```
echo "Enter your key vault name:" &&
read keyVaultName &&
az keyvault key list --vault-name $keyVaultName &&
echo "Press [ENTER] to continue ..."
```

Creating key using ARM template is different from creating key via data plane

Creating a key via ARM

- It's only possible to create *new* keys. It isn't possible to update existing keys, nor create new versions of existing keys. If the key already exists, then the existing key is retrieved from storage and used (no write operations will occur).
- To be authorized to use this API, the caller needs to have the "**Microsoft.KeyVault/vaults/keys/write**" role-based access control (RBAC) Action. The built-in "Key Vault Contributor" role is sufficient, since it authorizes all RBAC Actions that match the pattern "Microsoft.KeyVault/*".





Existing API (creating key via data plane)

- It's possible to create new keys, update existing keys, and create new versions of existing keys.
- The caller must be authorized to use this API. If the vault uses access policies, the caller must have "create" key permission; if the vault is enabled for RBAC, the caller must have "Microsoft.KeyVault/vaults/keys/create/action" RBAC DataAction.

Clean up resources

Other Key Vault quickstarts and tutorials build upon this quickstart. If you plan to continue on to work with subsequent quickstarts and tutorials, you may wish to leave these resources in place. When no longer needed, delete the resource group, which deletes the Key Vault and related resources. To delete the resource group by using Azure CLI or Azure PowerShell:

CLI

Azure CLI

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName &&
echo "Press [ENTER] to continue ..."
```

Next steps

In this quickstart, you created a key vault and a key using an ARM template, and validated the deployment. To learn more about Key Vault and Azure Resource Manager, see these articles.

- Read an Overview of Azure Key Vault
- Learn more about Azure Resource Manager
- Review the Key Vault security overview

Additional resources

Documentation

Microsoft.KeyVault/managedHSMs - Bicep, ARM template & Terraform AzAPI reference

Azure Microsoft.KeyVault/managedHSMs syntax and properties to use in Azure Resource Manager templates for deploying the resource. API version latest

Microsoft.KeyVault/vaults/keys - Bicep, ARM template & Terraform AzAPI reference

Azure Microsoft.KeyVault/vaults/keys syntax and properties to use in Azure Resource Manager templates for deploying the resource. API version latest

Azure Resource Manager template reference for Microsoft.KeyVault - Bicep, ARM template & Terraform AzAPI reference

Azure Resource Manager template reference for the Microsoft.KeyVault resource provider. Includes all resource types and versions.

Azure Quickstart - Create an Azure key vault and a secret by using Azure Resource Manager template

Quickstart showing how to create Azure key vaults, and add secrets to the vaults by using Azure Resource Manager template.

Microsoft.KeyVault/vaults/keys/versions - Bicep, ARM template & Terraform AzAPI reference

Azure Microsoft.KeyVault/vaults/keys/versions syntax and properties to use in Azure Resource Manager templates for deploying the resource. API version latest

Microsoft.KeyVault/vaults/privateEndpointConnections - Bicep, ARM template & Terraform AzAPI reference

Azure Microsoft.KeyVault/vaults/privateEndpointConnections syntax and properties to use in Azure Resource Manager templates for deploying the resource. API version latest

Microsoft.KeyVault/vaults/secrets 2022-07-01 - Bicep, ARM template & Terraform AzAPI reference

Azure Microsoft.KeyVault/vaults/secrets syntax and properties to use in Azure Resource Manager templates for deploying the resource. API version 2022-07-01

Azure Quickstart - Create an Azure key vault and a key by using Bicep

Quickstart showing how to create Azure key vaults, and add key to the vaults by using Bicep.

Show 5 more

Training

Module

Deploy and secure Azure Key Vault - Training

Protect your keys, certificates, and secrets in Azure Key Vault. Learn to configure key vault for the most secure deployment.

Quickstart: Create an Azure key vault and key using Terraform

Article • 04/14/2023

Azure Key Vault is a cloud service that provides a secure store for secrets, such as keys, passwords, and certificate. This article focuses on the process of deploying a Terraform file to create a key vault and a key.

Terraform enables the definition, preview, and deployment of cloud infrastructure. Using Terraform, you create configuration files using HCL syntax. The HCL syntax allows you to specify the cloud provider - such as Azure - and the elements that make up your cloud infrastructure. After you create your configuration files, you create an *execution plan* that allows you to preview your infrastructure changes before they're deployed. Once you verify the changes, you apply the execution plan to deploy the infrastructure.

In this article, you learn how to:

- ✓ Create a random value for the Azure resource group name using random_pet
- ✓ Create an Azure resource group using azurerm_resource_group
- ✓ Create a random value using random_string
- ✓ Create an Azure key vault using azurerm_key_vault
- ✓ Create an Azure key vault key using azurerm_key_vault_key

> ⓘ Note
>
> This article was partially created with the help of artificial intelligence. Before publishing, an author reviewed and revised the content as needed. See **Our principles for using AI-generated content in Microsoft Learn**.

Prerequisites

- Install and configure Terraform

Implement the Terraform code

> ⓘ Note

The sample code for this article is located in the **Azure Terraform GitHub repo**. You can view the log file containing the **test results from current and previous versions of Terraform**.

See more **articles and sample code showing how to use Terraform to manage Azure resources**

1. Create a directory in which to test and run the sample Terraform code and make it the current directory.

2. Create a file named `providers.tf` and insert the following code:

Terraform

```
terraform {
  required_version = ">=1.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
  }
}
provider "azurerm" {
  features {}
}
```

3. Create a file named `main.tf` and insert the following code:

Terraform

```
resource "random_pet" "rg_name" {
  prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
  name     = random_pet.rg_name.id
  location = var.resource_group_location
}

data "azurerm_client_config" "current" {}

resource "random_string" "azurerm_key_vault_name" {
  length  = 13
  lower   = true
```

```
  numeric = false
  special = false
  upper   = false
}

locals {
  current_user_id = coalesce(var.msi_id,
data.azurerm_client_config.current.object_id)
}

resource "azurerm_key_vault" "vault" {
  name                        = coalesce(var.vault_name,
"vault-${random_string.azurerm_key_vault_name.result}")
  location                    = azurerm_resource_group.rg.location
  resource_group_name         = azurerm_resource_group.rg.name
  tenant_id                   =
data.azurerm_client_config.current.tenant_id
  sku_name                    = var.sku_name
  soft_delete_retention_days = 7

  access_policy {
    tenant_id = data.azurerm_client_config.current.tenant_id
    object_id = local.current_user_id

    key_permissions    = var.key_permissions
    secret_permissions = var.secret_permissions
  }
}

resource "random_string" "azurerm_key_vault_key_name" {
  length  = 13
  lower   = true
  numeric = false
  special = false
  upper   = false
}

resource "azurerm_key_vault_key" "key" {
  name = coalesce(var.key_name,
"key-${random_string.azurerm_key_vault_key_name.result}")

  key_vault_id = azurerm_key_vault.vault.id
  key_type     = var.key_type
  key_size     = var.key_size
  key_opts     = var.key_ops

  rotation_policy {
    automatic {
      time_before_expiry = "P30D"
    }

    expire_after         = "P90D"
    notify_before_expiry = "P29D"
  }
}
```

4. Create a file named `variables.tf` and insert the following code:

Terraform

```
variable "resource_group_location" {
  type        = string
  description = "Location for all resources."
  default     = "eastus"
}

variable "resource_group_name_prefix" {
  type        = string
  description = "Prefix of the resource group name that's combined with
a random ID so name is unique in your Azure subscription."
  default     = "rg"
}

variable "vault_name" {
  type        = string
  description = "The name of the key vault to be created. The value
will be randomly generated if blank."
  default     = ""
}

variable "key_name" {
  type        = string
  description = "The name of the key to be created. The value will be
randomly generated if blank."
  default     = ""
}

variable "sku_name" {
  type        = string
  description = "The SKU of the vault to be created."
  default     = "standard"
  validation {
    condition     = contains(["standard", "premium"], var.sku_name)
    error_message = "The sku_name must be one of the following:
standard, premium."
  }
}

variable "key_permissions" {
  type        = list(string)
  description = "List of key permissions."
  default     = ["List", "Create", "Delete", "Get", "Purge", "Recover",
"Update", "GetRotationPolicy", "SetRotationPolicy"]
}

variable "secret_permissions" {
  type        = list(string)
  description = "List of secret permissions."
```

```
  default     = ["Set"]
}

variable "key_type" {
  description = "The JsonWebKeyType of the key to be created."
  default     = "RSA"
  type        = string
  validation {
    condition     = contains(["EC", "EC-HSM", "RSA", "RSA-HSM"],
var.key_type)
    error_message = "The key_type must be one of the following: EC, EC-
HSM, RSA, RSA-HSM."
  }
}

variable "key_ops" {
  type        = list(string)
  description = "The permitted JSON web key operations of the key to be
created."
  default     = ["decrypt", "encrypt", "sign", "unwrapKey", "verify",
"wrapKey"]
}

variable "key_size" {
  type        = number
  description = "The size in bits of the key to be created."
  default     = 2048
}

variable "msi_id" {
  type        = string
  description = "The Managed Service Identity ID. If this value isn't
null (the default), 'data.azurerm_client_config.current.object_id' will
be set to this value."
  default     = null
}
```

5. Create a file named `outputs.tf` and insert the following code:

Terraform

```
output "resource_group_name" {
  value = azurerm_resource_group.rg.name
}

output "azurerm_key_vault_name" {
  value = azurerm_key_vault.vault.name
}

output "azurerm_key_vault_id" {
  value = azurerm_key_vault.vault.id
}
```

Initialize Terraform

Run terraform init to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

```
Console

terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run terraform plan to create an execution plan.

```
Console

terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section.

Apply a Terraform execution plan

Run terraform apply to apply the execution plan to your cloud infrastructure.

```
Console

terraform apply main.tfplan
```

Key points:

- The `terraform apply` command above assumes you previously ran `terraform plan -out main.tfplan`.
- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

Verify the results

Azure CLI

1. Get the Azure key vault name.

   Console

   ```
   azurerm_key_vault_name=$(terraform output -raw
   azurerm_key_vault_name)
   ```

2. Run az keyvault key list to display information about the key vault's keys.

   Azure CLI

   ```
   az keyvault key list --vault-name $azurerm_key_vault_name
   ```

Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run terraform plan and specify the `destroy` flag.

   Console

   ```
   terraform plan -destroy -out main.destroy.tfplan
   ```

   Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section.

2. Run terraform apply to apply the execution plan.

```
Console

terraform apply main.destroy.tfplan
```

Troubleshoot Terraform on Azure

Troubleshoot common problems when using Terraform on Azure

Next steps

Key Vault security overview

Import HSM-protected keys to Key Vault

Article • 01/25/2023 • 2 minutes to read

For added assurance, when you use Azure Key Vault, you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. This scenario is often referred to as *bring your own key*, or BYOK. Azure Key Vault uses nCipher nShield family of HSMs (FIPS 140-2 Level 2 validated) to protect your keys.

This functionality is not available for Azure China 21Vianet.

> **Note**
>
> For more information about Azure Key Vault, see **What is Azure Key Vault?**
> For a getting started tutorial, which includes creating a key vault for HSM-protected keys, see **What is Azure Key Vault?**.

Supported HSMs

Transferring HSM-protected keys to Key Vault is supported via two different methods depending on the HSMs you use. Use this table to determine which method should be used for your HSMs to generate, and then transfer your own HSM-protected keys to use with Azure Key Vault.

Vendor Name	Vendor Type	Supported HSM models	Supported HSM-key transfer method
Cryptomathic	ISV (Enterprise Key Management System)	Multiple HSM brands and models including • nCipher • Thales • Utimaco See Cryptomathic site for details	Use new BYOK method
Entrust	Manufacturer, HSM as a Service	• nShield family of HSMs • nShield as a service	Use new BYOK method

Vendor Name	Vendor Type	Supported HSM models	Supported HSM-key transfer method
Fortanix	Manufacturer, HSM as a Service	• Self-Defending Key Management Service (SDKMS) • Equinix SmartKey	Use new BYOK method
IBM	Manufacturer	IBM 476x, CryptoExpress	Use new BYOK method
Marvell	Manufacturer	All LiquidSecurity HSMs with • Firmware version 2.0.4 or later • Firmware version 3.2 or newer	Use new BYOK method
nCipher	Manufacturer, HSM as a Service	• nShield family of HSMs • nShield as a service	**Method 1:** nCipher BYOK (deprecated). This method will not be supported after **June 30, 2021** **Method 2:** Use new BYOK method (recommended) See the Entrust row.
Securosys SA	Manufacturer, HSM as a service	Primus HSM family, Securosys Clouds HSM	Use new BYOK method
StorMagic	ISV (Enterprise Key Management System)	Multiple HSM brands and models including • Utimaco • Thales • nCipher See StorMagic site for details	Use new BYOK method
Thales	Manufacturer	• Luna HSM 7 family with firmware version 7.3 or newer	Use new BYOK method
Utimaco	Manufacturer, HSM as a service	u.trust Anchor, CryptoServer	Use new BYOK method

Next steps

- Review the Key Vault security overview to ensure security, durability and monitoring for your keys.
- Refer to BYOK specification for a complete description of the new BYOK method

Additional resources

Documentation

How to generate & transfer HSM-protected keys – BYOK – Azure Key Vault

Use this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Azure Key Vault. Also known as bring your own key (BYOK).

Best practices using Azure Key Vault Managed HSM

This document explains some of the best practices to use Key Vault

Control your data with Managed HSM

An overview of the safeguards and technical measures that help customers meet compliance

Enable multi-region replication on Azure Managed HSM (Preview)

Enable Multi-Region Replication on Azure Managed HSM (Preview)

Full backup/restore and selective restore for Azure Managed HSM

This document explains full backup/restore and selective restore

Azure Managed HSM Overview - Azure Managed HSM

Azure Managed HSM is a cloud service that safeguards your cryptographic keys for cloud applications.

How to generate and transfer HSM-protected keys for Azure Key Vault Managed HSM - Azure Key Vault

Use this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Managed HSM. Also known as bring your own key (BYOK).

Azure Managed HSM access control

Manage access permissions for Azure Managed HSM and keys. Covers the authentication and authorization model for Managed HSM, and how to secure your HSMs.

Show 5 more

Import HSM-protected keys to Key Vault (BYOK)

Article • 03/14/2023 • 7 minutes to read

For added assurance when you use Azure Key Vault, you can import or generate a key in a hardware security module (HSM); the key will never leave the HSM boundary. This scenario often is referred to as *bring your own key* (BYOK). Key Vault uses the nCipher nShield family of HSMs (FIPS 140-2 Level 2 validated) to protect your keys.

Use the information in this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Azure Key Vault.

> **Note**
>
> This functionality is not available for Azure China 21Vianet.
>
> This import method is available only for **supported HSMs**.

For more information, and for a tutorial to get started using Key Vault (including how to create a key vault for HSM-protected keys), see What is Azure Key Vault?.

Overview

Here's an overview of the process. Specific steps to complete are described later in the article.

- In Key Vault, generate a key (referred to as a *Key Exchange Key* (KEK)). The KEK must be an RSA-HSM key that has only the `import` key operation. Only Key Vault Premium and Managed HSM support RSA-HSM keys.
- Download the KEK public key as a .pem file.
- Transfer the KEK public key to an offline computer that is connected to an on-premises HSM.
- In the offline computer, use the BYOK tool provided by your HSM vendor to create a BYOK file.
- The target key is encrypted with a KEK, which stays encrypted until it is transferred to the Key Vault HSM. Only the encrypted version of your key leaves the on-premises HSM.
- A KEK that's generated inside a Key Vault HSM is not exportable. HSMs enforce the rule that no clear version of a KEK exists outside a Key Vault HSM.

- The KEK must be in the same key vault where the target key will be imported.
- When the BYOK file is uploaded to Key Vault, a Key Vault HSM uses the KEK private key to decrypt the target key material and import it as an HSM key. This operation happens entirely inside a Key Vault HSM. The target key always remains in the HSM protection boundary.

Prerequisites

The following table lists prerequisites for using BYOK in Azure Key Vault:

Requirement	More information
An Azure subscription	To create a key vault in Azure Key Vault, you need an Azure subscription. Sign up for a free trial.
A Key Vault Premium or Managed HSM to import HSM-protected keys	For more information about the service tiers and capabilities in Azure Key Vault, see Key Vault Pricing.
An HSM from the supported HSMs list and a BYOK tool and instructions provided by your HSM vendor	You must have permissions for an HSM and basic knowledge of how to use your HSM. See Supported HSMs.
Azure CLI version 2.1.0 or later	See Install the Azure CLI.

Supported HSMs

Vendor name	Vendor Type	Supported HSM models	More information
Cryptomathic	ISV (Enterprise Key Management System)	Multiple HSM brands and models including • nCipher • Thales • Utimaco See Cryptomathic site for details	Cryptomathic BYOK tool and documentation
Entrust	Manufacturer, HSM as a service	• nShield family of HSMs • nShield as a service	nCipher new BYOK tool and documentation

Vendor name	Vendor Type	Supported HSM models	More information
Fortanix	Manufacturer, HSM as a service	• Self-Defending Key Management Service (SDKMS) • Equinix SmartKey	Exporting SDKMS keys to Cloud Providers for BYOK - Azure Key Vault
IBM	Manufacturer	IBM 476x, CryptoExpress	IBM Enterprise Key Management Foundation
Marvell	Manufacturer	All LiquidSecurity HSMs with • Firmware version 2.0.4 or later • Firmware version 3.2 or newer	Marvell BYOK tool and documentation
nCipher	Manufacturer, HSM as a service	• nShield family of HSMs • nShield as a service	nCipher new BYOK tool and documentation
Securosys SA	Manufacturer, HSM as a service	Primus HSM family, Securosys Clouds HSM	Primus BYOK tool and documentation
StorMagic	ISV (Enterprise Key Management System)	Multiple HSM brands and models including • Utimaco • Thales • nCipher See StorMagic site for details	SvKMS and Azure Key Vault BYOK
Thales	Manufacturer	• Luna HSM 7 family with firmware version 7.3 or newer	Luna BYOK tool and documentation
Utimaco	Manufacturer, HSM as a service	u.trust Anchor, CryptoServer	Utimaco BYOK tool and Integration guide

Supported key types

Key name	Key type	Key size/curve	Origin	Description

Key name	Key type	Key size/curve	Origin	Description
Key Exchange Key (KEK)	RSA	2,048-bit 3,072-bit 4,096-bit	Azure Key Vault HSM	An HSM-backed RSA key pair generated in Azure Key Vault
Target key				
	RSA	2,048-bit 3,072-bit 4,096-bit	Vendor HSM	The key to be transferred to the Azure Key Vault HSM
	EC	P-256 P-384 P-521	Vendor HSM	The key to be transferred to the Azure Key Vault HSM

Generate and transfer your key to Key Vault Premium HSM or Managed HSM

To generate and transfer your key to a Key Vault Premium or Managed HSM:

- Step 1: Generate a KEK
- Step 2: Download the KEK public key
- Step 3: Generate and prepare your key for transfer
- Step 4: Transfer your key to Azure Key Vault

Generate a KEK

A KEK is an RSA key that's generated in a Key Vault Premium or Managed HSM. The KEK is used to encrypt the key you want to import (the *target* key).

The KEK must be:

- An RSA-HSM key (2,048-bit; 3,072-bit; or 4,096-bit)
- Generated in the same key vault where you intend to import the target key
- Created with allowed key operations set to `import`

> **Note**
>
> The KEK must have 'import' as the only allowed key operation. 'import' is mutually exclusive with all other key operations.

Use the az keyvault key create command to create a KEK that has key operations set to `import`. Record the key identifier (`kid`) that's returned from the following command. (You will use the `kid` value in Step 3.)

Azure CLI

Azure CLI

```
az keyvault key create --kty RSA-HSM --size 4096 --name KEKforBYOK --ops
import --vault-name ContosoKeyVaultHSM
```

For Managed HSM:

Azure CLI

```
az keyvault key create --kty RSA-HSM --size 4096 --name KEKforBYOK --ops
import --hsm-name ContosoKeyVaultHSM
```

Download the KEK public key

Use az keyvault key download to download the KEK public key to a .pem file. The target key you import is encrypted by using the KEK public key.

Azure CLI

Azure CLI

```
az keyvault key download --name KEKforBYOK --vault-name
ContosoKeyVaultHSM --file KEKforBYOK.publickey.pem
```

For Managed HSM:

Azure CLI

```
az keyvault key download --name KEKforBYOK --hsm-name ContosoKeyVaultHSM
--file KEKforBYOK.publickey.pem
```

Transfer the KEKforBYOK.publickey.pem file to your offline computer. You will need this file in the next step.

Generate and prepare your key for transfer

Refer to your HSM vendor's documentation to download and install the BYOK tool. Follow instructions from your HSM vendor to generate a target key, and then create a key transfer package (a BYOK file). The BYOK tool will use the `kid` from Step 1 and the KEKforBYOK.publickey.pem file you downloaded in Step 2 to generate an encrypted target key in a BYOK file.

Transfer the BYOK file to your connected computer.

> **Note**
>
> Importing RSA 1,024-bit keys is not supported. Importing Elliptic Curve key with curve P-256K is not supported.
>
> **Known issue**: Importing an RSA 4K target key from Luna HSMs is only supported with firmware 7.4.0 or newer.

Transfer your key to Azure Key Vault

To complete the key import, transfer the key transfer package (a BYOK file) from your disconnected computer to the internet-connected computer. Use the az keyvault key import command to upload the BYOK file to the Key Vault HSM.

To import an RSA key use following command. Parameter --kty is optional and defaults to 'RSA-HSM'.

Azure CLI

Azure CLI

```
az keyvault key import --vault-name ContosoKeyVaultHSM --name
ContosoFirstHSMkey --byok-file KeyTransferPackage-
ContosoFirstHSMkey.byok
```

For Managed HSM

Azure CLI

```
az keyvault key import --hsm-name ContosoKeyVaultHSM --name
ContosoFirstHSMkey --byok-file KeyTransferPackage-
ContosoFirstHSMkey.byok
```

To import an EC key, you must specify key type and the curve name.

Azure CLI

Azure CLI

```
az keyvault key import --vault-name ContosoKeyVaultHSM --name
ContosoFirstHSMkey --kty EC-HSM --curve-name "P-256" --byok-file
KeyTransferPackage-ContosoFirstHSMkey.byok
```

For Managed HSM

Azure CLI

```
az keyvault key import --hsm-name ContosoKeyVaultHSM --name
ContosoFirstHSMkey --byok-file --kty EC-HSM --curve-name "P-256"
KeyTransferPackage-ContosoFirstHSMkey.byok
```

If the upload is successful, Azure CLI displays the properties of the imported key.

Next steps

You can now use this HSM-protected key in your key vault. For more information, see this price and feature comparison.

Additional resources

Documentation

Best practices using Azure Key Vault Managed HSM

This document explains some of the best practices to use Key Vault

Azure Managed HSM logging

Use this tutorial to help you get started with Managed HSM logging.

How to generate & transfer HSM-protected keys – Azure Key Vault

Learn how to plan for, generate, and then transfer your own HSM-protected keys to use with Azure Key Vault. Also known as BYOK or bring your own key.

Azure Managed HSM access control

Manage access permissions for Azure Managed HSM and keys. Covers the authentication and authorization model for Managed HSM, and how to secure your HSMs.

Configure key auto-rotation in Azure Key Vault Managed HSM

Use this guide to learn how to configure automated the rotation of a key in Azure Key Vault Managed HSM

Enable multi-region replication on Azure Managed HSM (Preview)

Enable Multi-Region Replication on Azure Managed HSM (Preview)

Azure Managed HSM Overview - Azure Managed HSM

Azure Managed HSM is a cloud service that safeguards your cryptographic keys for cloud applications.

Manage keys in a managed HSM - Azure Key Vault

Use this article to manage keys in a managed HSM

Show 5 more

Import HSM-protected keys for Key Vault (nCipher)

Article • 01/25/2023 • 17 minutes to read

> ⚠ **Warning**
>
> The HSM-key import method described in this document is **deprecated** and will not be supported after June 30, 2021. It only works with nCipher nShield family of HSMs with firmware 12.40.2 or newer. Using **new method to import HSM-keys** is strongly recommended.

> ⓘ **Note**
>
> We recommend that you use the Azure Az PowerShell module to interact with Azure. See **Install Azure PowerShell** to get started. To learn how to migrate to the Az PowerShell module, see **Migrate Azure PowerShell from AzureRM to Az**.

For added assurance, when you use Azure Key Vault, you can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary. This scenario is often referred to as *bring your own key*, or BYOK. Azure Key Vault uses nCipher nShield family of HSMs (FIPS 140-2 Level 2 validated) to protect your keys.

Use this article to help you plan for, generate, and then transfer your own HSM-protected keys to use with Azure Key Vault.

This functionality isn't available for Azure China 21Vianet.

> ⓘ **Note**
>
> For more information about Azure Key Vault, see **What is Azure Key Vault?** For a getting started tutorial, which includes creating a key vault for HSM-protected keys, see **What is Azure Key Vault?**.

More information about generating and transferring an HSM-protected key over the Internet:

- You generate the key from an offline workstation, which reduces the attack surface.
- The key is encrypted with a Key Exchange Key (KEK), which stays encrypted until it's transferred to the Azure Key Vault HSMs. Only the encrypted version of your key

leaves the original workstation.

- The toolset sets properties on your tenant key that binds your key to the Azure Key Vault security world. So after the Azure Key Vault HSMs receive and decrypt your key, only these HSMs can use it. Your key cannot be exported. This binding is enforced by the nCipher HSMs.
- The Key Exchange Key (KEK) that is used to encrypt your key is generated inside the Azure Key Vault HSMs and is not exportable. The HSMs enforce that there can be no clear version of the KEK outside the HSMs. In addition, the toolset includes attestation from nCipher that the KEK is not exportable and was generated inside a genuine HSM that was manufactured by nCipher.
- The toolset includes attestation from nCipher that the Azure Key Vault security world was also generated on a genuine HSM manufactured by nCipher. This attestation demonstrates that Microsoft is using genuine hardware.
- Microsoft uses separate KEKs and separate Security Worlds in each geographical region. This separation ensures that your key can be used only in data centers in the region in which you encrypted it. For example, a key from a European customer cannot be used in data centers in North American or Asia.

More information about nCipher HSMs and Microsoft services

nCipher Security, an Entrust Datacard company, is a leader in the general purpose HSM market, empowering world-leading organizations by delivering trust, integrity and control to their business critical information and applications. nCipher's cryptographic solutions secure emerging technologies – cloud, IoT, blockchain, digital payments – and help meet new compliance mandates, using the same proven technology that global organizations depend on today to protect against threats to their sensitive data, network communications and enterprise infrastructure. nCipher delivers trust for business critical applications, ensuring the integrity of data and putting customers in complete control – today, tomorrow, always.

Microsoft has collaborated with nCipher Security to enhance the state of art for HSMs. These enhancements enable you to get the typical benefits of hosted services without relinquishing control over your keys. Specifically, these enhancements let Microsoft manage the HSMs so that you do not have to. As a cloud service, Azure Key Vault scales up at short notice to meet your organization's usage spikes. At the same time, your key is protected inside Microsoft's HSMs: You retain control over the key lifecycle because you generate the key and transfer it to Microsoft's HSMs.

Implementing bring your own key (BYOK) for Azure Key Vault

Use the following information and procedures if you will generate your own HSM-protected key and then transfer it to Azure Key Vault. This is known as the Bring Your Own Key (BYOK) scenario.

Prerequisites for BYOK

See the following table for a list of prerequisites for bring your own key (BYOK) for Azure Key Vault.

Requirement	More information
A subscription to Azure	To create an Azure Key Vault, you need an Azure subscription: Sign up for free trial
The Azure Key Vault Premium service tier to support HSM-protected keys	For more information about the service tiers and capabilities for Azure Key Vault, see the Azure Key Vault Pricing website.
nCipher nShield HSMs, smartcards, and support software	You must have access to a nCipher Hardware Security Module and basic operational knowledge of nCipher nShield HSMs. See nCipher nShield Hardware Security Module for the list of compatible models, or to purchase an HSM if you do not have one.

Requirement	More information
The following hardware and software: 1. An offline x64 workstation with a minimum Windows operation system of Windows 7 and nCipher nShield software that is at least version 11.50. If this workstation runs Windows 7, you must install Microsoft .NET Framework 4.5 ↗. 2. A workstation that is connected to the Internet and has a minimum Windows operating system of Windows 7 and Azure PowerShell **minimum version 1.1.0** installed. 3. A USB drive or other portable storage device that has at least 16-MB free space.	For security reasons, we recommend that the first workstation is not connected to a network. However, this recommendation is not programmatically enforced. In the instructions that follow, this workstation is referred to as the disconnected workstation. In addition, if your tenant key is for a production network, we recommend that you use a second, separate workstation to download the toolset, and upload the tenant key. But for testing purposes, you can use the same workstation as the first one. In the instructions that follow, this second workstation is referred to as the Internet-connected workstation.

Generate and transfer your key to Azure Key Vault HSM

You'll use the following five steps to generate and transfer your key to an Azure Key Vault HSM:

- Step 1: Prepare your Internet-connected workstation
- Step 2: Prepare your disconnected workstation
- Step 3: Generate your key
- Step 4: Prepare your key for transfer
- Step 5: Transfer your key to Azure Key Vault

Prepare your Internet-connected workstation

For this first step, do the following procedures on your workstation that is connected to the Internet.

Install Azure PowerShell

From the Internet-connected workstation, download and install the Azure PowerShell module that includes the cmdlets to manage Azure Key Vault. For installation instructions, see How to install and configure Azure PowerShell.

Get your Azure subscription ID

Start an Azure PowerShell session and sign in to your Azure account by using the following command:

```
PowerShell

    Connect-AzAccount
```

In the pop-up browser window, enter your Azure account user name and password. Then, use the Get-AzSubscription command:

```
PowerShell

    Get-AzSubscription
```

From the output, locate the ID for the subscription you will use for Azure Key Vault. You'll need this subscription ID later.

Do not close the Azure PowerShell window.

Download the BYOK toolset for Azure Key Vault

Go to the Microsoft Download Center and download the Azure Key Vault BYOK toolset for your geographic region or instance of Azure. Use the following information to identify the package name to download and its corresponding SHA-256 package hash:

United States:

KeyVault-BYOK-Tools-UnitedStates.zip

2E8C00320400430106366A4E8C67B79015524E4EC24A2D3A6DC513CA1823B0D4

Europe:

KeyVault-BYOK-Tools-Europe.zip

9AAA63E2E7F20CF9BB62485868754203721D2F88D300910634A32DFA1FB19E4A

Asia:

KeyVault-BYOK-Tools-AsiaPacific.zip

4BC14059BF0FEC562CA927AF621DF665328F8A13616F44C977388EC7121EF6B5

Latin America:

KeyVault-BYOK-Tools-LatinAmerica.zip

E7DFAFF579AFE1B9732C30D6FD80C4D03756642F25A538922DD1B01A4FACB619

Japan:

KeyVault-BYOK-Tools-Japan.zip

3933C13CC6DC06651295ADC482B027AF923A76F1F6BF98B4D4B8E94632DEC7DF

Korea:

KeyVault-BYOK-Tools-Korea.zip

71AB6BCFE06950097C8C18D532A9184BEF52A74BB944B8610DDDA05344ED136F

South Africa:

KeyVault-BYOK-Tools-SouthAfrica.zip

C41060C5C0170AAAAD896DA732E31433D14CB9FC83AC3C67766F46D98620784A

UAE:

KeyVault-BYOK-Tools-UAE.zip

FADE80210B06962AA0913EA411DAB977929248C65F365FD953BB9F241D5FC0D3

Australia:

KeyVault-BYOK-Tools-Australia.zip

CD0FB7365053DEF8C35116D7C92D203C64A3D3EE2452A025223EEB166901C40A

Azure Government:

KeyVault-BYOK-Tools-USGovCloud.zip

F8DB2FC914A7360650922391D9AA79FF030FD3048B5795EC83ADC59DB018621A

US Government DOD:

KeyVault-BYOK-Tools-USGovernmentDoD.zip

A79DD8C6DFFF1B00B91D1812280207A205442B3DDF861B79B8B991BB55C35263

Canada:

KeyVault-BYOK-Tools-Canada.zip

61BE1A1F80AC79912A42DEBBCC42CF87C88C2CE249E27193463088579971 7C7B

Germany:

KeyVault-BYOK-Tools-Germany.zip

5385E615880AAFC02AFD9841F7BADD025D7EE819894AA29ED3C71C3F844C45D6

Germany Public:

KeyVault-BYOK-Tools-Germany-Public.zip

54534936D0A0C99C8117DB724C34A5E50FD204CFCBD75C78972B785865364A29

India:

KeyVault-BYOK-Tools-India.zip

49EDCEB3091CF1DF7B156D5B495A4ADE1CFBA77641134F61B0E0940121C436C8

France:

KeyVault-BYOK-Tools-France.zip

5C9D1F3E4125B0C09E9F60897C9AE3A8B4CB0E7D13A14F3EDBD280128F8FE7DF

United Kingdom:

KeyVault-BYOK-Tools-UnitedKingdom.zip

432746BD0D3176B708672CCFF19D6144FCAA9E5EB29BB056489D3782B3B80849

Switzerland:

KeyVault-BYOK-Tools-Switzerland.zip

88CF8D39899E26D456D4E0BC57E5C94913ABF1D73A89013FCE3BBD9599AD2FE9

To validate the integrity of your downloaded BYOK toolset, from your Azure PowerShell session, use the Get-FileHash cmdlet.

```
PowerShell
```

```
Get-FileHash KeyVault-BYOK-Tools-*.zip
```

The toolset includes:

- A Key Exchange Key (KEK) package that has a name beginning with **BYOK-KEK-pkg-**.
- A Security World package that has a name beginning with **BYOK-SecurityWorld-pkg-**.
- A Python script named **verifykeypackage.py.**
- A command-line executable file named **KeyTransferRemote.exe** and associated DLLs.
- A Visual C++ Redistributable Package, named **vcredist_x64.exe.**

Copy the package to a USB drive or other portable storage.

Prepare your disconnected workstation

For this second step, do the following procedures on the workstation that is not connected to a network (either the Internet or your internal network).

Prepare the disconnected workstation with nCipher nShield HSM

Install the nCipher support software on a Windows computer, and then attach a nCipher nShield HSM to that computer.

Ensure that the nCipher tools are in your path (**%nfast_home%\bin**). For example, type :

Windows Command Prompt

```
set PATH=%PATH%;"%nfast_home%\bin"
```

For more information, see the user guide included with the nShield HSM.

Install the BYOK toolset on the disconnected workstation

Copy the BYOK toolset package from the USB drive or other portable storage, and then:

1. Extract the files from the downloaded package into any folder.
2. From that folder, run vcredist_x64.exe.

3. Follow the instructions to the install the Visual C++ runtime components for Visual Studio 2013.

Generate your key

For this third step, do the following procedures on the disconnected workstation. To complete this step your HSM must be in initialization mode.

Change the HSM mode to 'I'

If you are using nCipher nShield Edge, to change the mode: 1. Use the Mode button to highlight the required mode. 2. Within a few seconds, press and hold the Clear button for a couple of seconds. If the mode changes, the new mode's LED stops flashing and remains lit. The Status LED might flash irregularly for a few seconds and then flashes regularly when the device is ready. Otherwise, the device remains in the current mode, with the appropriate mode LED lit.

Create a security world

Start a command prompt and run the nCipher new-world program.

Windows Command Prompt

```
 new-world.exe --initialize --cipher-suite=DLf3072s256mRijndael --module=1 --acs-quorum=2/3
```

This program creates a **Security World** file at %NFAST_KMDATA%\local\world, which corresponds to the C:\ProgramData\nCipher\Key Management Data\local folder. You can use different values for the quorum but in our example, you're prompted to enter three blank cards and pins for each one. Then, any two cards give full access to the security world. These cards become the **Administrator Card Set** for the new security world.

> ⓘ **Note**
>
> If your HSM does not support the newer cypher suite DLf3072s256mRijndael, you can replace `--cipher-suite= DLf3072s256mRijndael` with `--cipher-suite=DLf1024s160mRijndael`.
>
> Security world created with new-world.exe that ships with nCipher software version 12.50 is not compatible with this BYOK procedure. There are two options available:

1. Downgrade nCipher software version to 12.40.2 to create a new security world.
2. Contact nCipher support and request them to provide a hotfix for 12.50 software version, which allows you to use 12.40.2 version of new-world.exe that is compatible with this BYOK procedure.

Then:

- Back up the world file. Secure and protect the world file, the Administrator Cards, and their pins, and make sure that no single person has access to more than one card.

Change the HSM mode to 'O'

If you are using nCipher nShield Edge, to change the mode: 1. Use the Mode button to highlight the required mode. 2. Within a few seconds, press and hold the Clear button for a couple of seconds. If the mode changes, the new mode's LED stops flashing and remains lit. The Status LED might flash irregularly for a few seconds and then flashes regularly when the device is ready. Otherwise, the device remains in the current mode, with the appropriate mode LED lit.

Validate the downloaded package

This step is optional but recommended so that you can validate the following:

- The Key Exchange Key that is included in the toolset has been generated from a genuine nCipher nShield HSM.
- The hash of the Security World that is included in the toolset has been generated in a genuine nCipher nShield HSM.
- The Key Exchange Key is non-exportable.

> ⓘ **Note**
>
> To validate the downloaded package, the HSM must be connected, powered on, and must have a security world on it (such as the one you've just created).

To validate the downloaded package:

1. Run the verifykeypackage.py script by typing one of the following, depending on your geographic region or instance of Azure:

- For North America:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-NA-1 -w BYOK-SecurityWorld-pkg-NA-1
```

- For Europe:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-EU-1 -w BYOK-SecurityWorld-pkg-EU-1
```

- For Asia:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-AP-1 -w BYOK-SecurityWorld-pkg-AP-1
```

- For Latin America:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-LATAM-1 -w BYOK-SecurityWorld-pkg-LATAM-1
```

- For Japan:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-JPN-1 -w BYOK-SecurityWorld-pkg-JPN-1
```

- For Korea:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-KOREA-1 -w BYOK-SecurityWorld-pkg-KOREA-1
```

- For South Africa:

```
Azure PowerShell
```

```
"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-SA-1 -w BYOK-SecurityWorld-pkg-SA-1
```

- For UAE:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-UAE-1 -w BYOK-SecurityWorld-pkg-UAE-1
```

- For Australia:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-AUS-1 -w BYOK-SecurityWorld-pkg-AUS-1
```

- For Azure Government, which uses the US government instance of Azure:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-USGOV-1 -w BYOK-SecurityWorld-pkg-USGOV-1
```

- For US Government DOD:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-USDOD-1 -w BYOK-SecurityWorld-pkg-USDOD-1
```

- For Canada:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-CANADA-1 -w BYOK-SecurityWorld-pkg-CANADA-1
```

- For Germany:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-GERMANY-1 -w BYOK-SecurityWorld-pkg-GERMANY-1
```

- For Germany Public:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-GERMANY-1 -w BYOK-SecurityWorld-pkg-GERMANY-1
```

- For India:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-INDIA-1 -w BYOK-SecurityWorld-pkg-INDIA-1
```

- For France:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-FRANCE-1 -w BYOK-SecurityWorld-pkg-FRANCE-1
```

- For United Kingdom:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-UK-1 -w BYOK-SecurityWorld-pkg-UK-1
```

- For Switzerland:

```
Azure PowerShell

"%nfast_home%\python\bin\python" verifykeypackage.py -k BYOK-KEK-pkg-SUI-1 -w BYOK-SecurityWorld-pkg-SUI-1
```

> **Tip**
>
> The nCipher nShield software includes Python at %NFAST_HOME%\python\bin

2. Confirm that you see the following, which indicates successful validation: **Result: SUCCESS**

This script validates the signer chain up to the nShield root key. The hash of this root key is embedded in the script and its value should be **59178a47 de508c3f 291277ee 184f46c4 f1d9c639**. You can also confirm this value separately by visiting the nCipher website.

You're now ready to create a new key.

Create a new key

Generate a key by using the nCipher nShield **generatekey** program.

Run the following command to generate the key:

Azure PowerShell

```
generatekey --generate simple type=RSA size=2048 protect=module
ident=contosokey plainname=contosokey nvram=no pubexp=
```

When you run this command, use these instructions:

- The parameter *protect* must be set to the value **module**, as shown. This creates a module-protected key. The BYOK toolset does not support OCS-protected keys.
- Replace the value of *contosokey* for the **ident** and **plainname** with any string value. To minimize administrative overheads and reduce the risk of errors, we recommend that you use the same value for both. The **ident** value must contain only numbers, dashes, and lower case letters.
- The pubexp is left blank (default) in this example, but you can specify specific values.

This command creates a Tokenized Key file in your %NFAST_KMDATA%\local folder with a name starting with **key_simple_**, followed by the **ident** that was specified in the command. For example: **key_simple_contosokey**. This file contains an encrypted key.

Back up this Tokenized Key File in a safe location.

> **Important**
>
> When you later transfer your key to Azure Key Vault, Microsoft cannot export this key back to you so it becomes extremely important that you back up your key and security world safely. Contact **nCipher** for guidance and best practices for backing up your key.

You are now ready to transfer your key to Azure Key Vault.

Prepare your key for transfer

For this fourth step, do the following procedures on the disconnected workstation.

Create a copy of your key with reduced permissions

Open a new command prompt and change the current directory to the location where you unzipped the BYOK zip file. To reduce the permissions on your key, from a command prompt, run one of the following, depending on your geographic region or instance of Azure:

- For North America:

  Azure PowerShell

  ```
  KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
  contosokey -ExchangeKeyPackage BYOK-KEK-pkg-NA-1 -
  NewSecurityWorldPackage BYOK-SecurityWorld-pkg-NA-
  ```

- For Europe:

  Azure PowerShell

  ```
  KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
  contosokey -ExchangeKeyPackage BYOK-KEK-pkg-EU-1 -
  NewSecurityWorldPackage BYOK-SecurityWorld-pkg-EU-1
  ```

- For Asia:

  Azure PowerShell

  ```
  KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
  contosokey -ExchangeKeyPackage BYOK-KEK-pkg-AP-1 -
  NewSecurityWorldPackage BYOK-SecurityWorld-pkg-AP-1
  ```

- For Latin America:

  Azure PowerShell

  ```
  KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
  contosokey -ExchangeKeyPackage BYOK-KEK-pkg-LATAM-1 -
  NewSecurityWorldPackage BYOK-SecurityWorld-pkg-LATAM-1
  ```

- For Japan:

Azure PowerShell

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-JPN-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-JPN-1
```

- For Korea:

Azure PowerShell

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-KOREA-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-KOREA-1
```

- For South Africa:

Azure PowerShell

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-SA-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-SA-1
```

- For UAE:

Azure PowerShell

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-UAE-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-UAE-1
```

- For Australia:

Azure PowerShell

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-AUS-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-AUS-1
```

- For Azure Government, which uses the US government instance of Azure:

Azure PowerShell

```
KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-USGOV-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-USGOV-1
```

- For US Government DOD:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-USDOD-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-USDOD-1
```

- For Canada:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-CANADA-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-CANADA-1
```

- For Germany:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-GERMANY-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-GERMANY-1
```

- For Germany Public:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-GERMANY-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-GERMANY-1
```

- For India:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-INDIA-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-INDIA-1
```

- For France:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-FRANCE-1 -
```

```
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-FRANCE-1
```

- For United Kingdom:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-UK-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-UK-1
```

- For Switzerland:

```
Azure PowerShell

KeyTransferRemote.exe -ModifyAcls -KeyAppName simple -KeyIdentifier
contosokey -ExchangeKeyPackage BYOK-KEK-pkg-SUI-1 -
NewSecurityWorldPackage BYOK-SecurityWorld-pkg-SUI-1
```

When you run this command, replace *contosokey* with the same value you specified in **Step 3.5: Create a new key** from the Generate your key step.

You are asked to plug in your security world admin cards.

When the command completes, you see **Result: SUCCESS** and the copy of your key with reduced permissions are in the file named key_xferacld_<contosokey>.

You may inspects the ACLS using following commands using the nCipher nShield utilities:

- aclprint.py:

```
Windows Command Prompt

"%nfast_home%\bin\preload.exe" -m 1 -A xferacld -K contosokey
"%nfast_home%\python\bin\python"
"%nfast_home%\python\examples\aclprint.py"
```

- kmfile-dump.exe:

```
Windows Command Prompt

"%nfast_home%\bin\kmfile-dump.exe"
"%NFAST_KMDATA%\local\key_xferacld_contosokey"
```

When you run these commands, replace contosokey with the same value you specified in **Step 3.5: Create a new key** from the Generate your key step.

Encrypt your key by using Microsoft's Key Exchange Key

Run one of the following commands, depending on your geographic region or instance of Azure:

- For North America:

```
Azure PowerShell

KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-NA-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-NA-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Europe:

```
Azure PowerShell

KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-EU-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-EU-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Asia:

```
Azure PowerShell

KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-AP-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-AP-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Latin America:

```
Azure PowerShell

KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-LATAM-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-LATAM-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For Japan:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-JPN-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-JPN-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Korea:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-KOREA-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-KOREA-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For South Africa:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-SA-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-SA-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For UAE:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-UAE-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-UAE-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Australia:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-AUS-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-AUS-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Azure Government, which uses the US government instance of Azure:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-USGOV-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-USGOV-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For US Government DOD:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-USDOD-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-USDOD-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For Canada:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-CANADA-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-CANADA-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For Germany:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-GERMANY-1 -NewSecurityWorldPackage
BYOK-SecurityWorld-pkg-GERMANY-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For Germany Public:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-GERMANY-1 -NewSecurityWorldPackage
BYOK-SecurityWorld-pkg-GERMANY-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For India:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-INDIA-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-INDIA-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For France:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-France-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-France-1 -SubscriptionId SubscriptionID -
KeyFriendlyName ContosoFirstHSMkey
```

- For United Kingdom:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-UK-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-UK-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

- For Switzerland:

Azure PowerShell

```
KeyTransferRemote.exe -Package -KeyIdentifier contosokey -
ExchangeKeyPackage BYOK-KEK-pkg-SUI-1 -NewSecurityWorldPackage BYOK-
SecurityWorld-pkg-SUI-1 -SubscriptionId SubscriptionID -KeyFriendlyName
ContosoFirstHSMkey
```

When you run this command, use these instructions:

- Replace *contosokey* with the identifier that you used to generate the key in **Step 3.5: Create a new key** from the Generate your key step.
- Replace *SubscriptionID* with the ID of the Azure subscription that contains your key vault. You retrieved this value previously, in **Step 1.2: Get your Azure subscription ID** from the Prepare your Internet-connected workstation step.
- Replace *ContosoFirstHSMKey* with a label that is used for your output file name.

When this completes successfully, it displays **Result: SUCCESS** and there is a new file in the current folder that has the following name: KeyTransferPackage-*ContosoFirstHSMkey*.byok

Copy your key transfer package to the Internet-connected workstation

Use a USB drive or other portable storage to copy the output file from the previous step (KeyTransferPackage-ContosoFirstHSMkey.byok) to your Internet-connected workstation.

Transfer your key to Azure Key Vault

For this final step, on the Internet-connected workstation, use the Add-AzKeyVaultKey cmdlet to upload the key transfer package that you copied from the disconnected workstation to the Azure Key Vault HSM:

PowerShell

```
    Add-AzKeyVaultKey -VaultName 'ContosoKeyVaultHSM' -Name
'ContosoFirstHSMkey' -KeyFilePath 'c:\KeyTransferPackage-
ContosoFirstHSMkey.byok' -Destination 'HSM'
```

If the upload is successful, you see displayed the properties of the key that you just added.

Next steps

You can now use this HSM-protected key in your key vault. For more information, see this price and feature comparison.

Additional resources

Documentation

How to generate & transfer HSM-protected keys – BYOK – Azure Key Vault

Use this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Azure Key Vault. Also known as bring your own key (BYOK).

About Azure Managed HSM security domain

Overview of the Managed HSM Security Domain, a set of artifacts needed to recover a Managed HSM

Secure access to a managed HSM - Azure Key Vault Managed HSM

Learn how to secure access to Managed HSM using Azure RBAC and Managed HSM local RBAC

How to generate and transfer HSM-protected keys for Azure Key Vault Managed HSM - Azure Key Vault

Use this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Managed HSM. Also known as bring your own key (BYOK).

Best practices using Azure Key Vault Managed HSM

This document explains some of the best practices to use Key Vault

Enable multi-region replication on Azure Managed HSM (Preview)

Enable Multi-Region Replication on Azure Managed HSM (Preview)

Manage keys in a managed HSM - Azure Key Vault

Use this article to manage keys in a managed HSM

How to generate & transfer HSM-protected keys – Azure Key Vault

Learn how to plan for, generate, and then transfer your own HSM-protected keys to use with Azure Key Vault. Also known as BYOK or bring your own key.

Show 5 more

Key types, algorithms, and operations

Article • 02/10/2023 • 10 minutes to read

Key Vault supports two resource types: vaults and managed HSMs. Both resources types support various encryption keys. To see a summary of supported key types, protection types by each resource type, see About keys.

Following table shows a summary of key types and supported algorithms.

Key types/sizes/curves	Encrypt/Decrypt (Wrap/Unwrap)	Sign/Verify
EC-P256, EC-P256K, EC-P384, EC-P521	NA	ES256 ES256K ES384 ES512
RSA 2K, 3K, 4K	RSA1_5 RSA-OAEP RSA-OAEP-256	PS256 PS384 PS512 RS256 RS384 RS512 RSNULL
AES 128-bit, 256-bit (Managed HSM only)	AES-KW AES-GCM AES-CBC	NA

EC algorithms

The following algorithm identifiers are supported with EC-HSM keys

Curve Types

- **P-256** - The NIST curve P-256, defined at DSS FIPS PUB 186-4.
- **P-256K** - The SEC curve SECP256K1, defined at SEC 2: Recommended Elliptic Curve Domain Parameters.
- **P-384** - The NIST curve P-384, defined at DSS FIPS PUB 186-4.
- **P-521** - The NIST curve P-521, defined at DSS FIPS PUB 186-4.

SIGN/VERIFY

- **ES256** - ECDSA for SHA-256 digests and keys created with curve P-256. This algorithm is described at RFC7518.
- **ES256K** - ECDSA for SHA-256 digests and keys created with curve P-256K. This algorithm is pending standardization.
- **ES384** - ECDSA for SHA-384 digests and keys created with curve P-384. This algorithm is described at RFC7518.
- **ES512** - ECDSA for SHA-512 digests and keys created with curve P-521. This algorithm is described at RFC7518.

RSA algorithms

The following algorithm identifiers are supported with RSA and RSA-HSM keys

WRAPKEY/UNWRAPKEY, ENCRYPT/DECRYPT

- **RSA1_5** - RSAES-PKCS1-V1_5 [RFC3447] key encryption
- **RSA-OAEP** - RSAES using Optimal Asymmetric Encryption Padding (OAEP) [RFC3447], with the default parameters specified by RFC 3447 in Section A.2.1. Those default parameters are using a hash function of SHA-1 and a mask generation function of MGF1 with SHA-1.
- **RSA-OAEP-256** – RSAES using Optimal Asymmetric Encryption Padding with a hash function of SHA-256 and a mask generation function of MGF1 with SHA-256

SIGN/VERIFY

- **PS256** - RSASSA-PSS using SHA-256 and MGF1 with SHA-256, as described in RFC7518.
- **PS384** - RSASSA-PSS using SHA-384 and MGF1 with SHA-384, as described in RFC7518.
- **PS512** - RSASSA-PSS using SHA-512 and MGF1 with SHA-512, as described in RFC7518.
- **RS256** - RSASSA-PKCS-v1_5 using SHA-256. The application supplied digest value must be computed using SHA-256 and must be 32 bytes in length.
- **RS384** - RSASSA-PKCS-v1_5 using SHA-384. The application supplied digest value must be computed using SHA-384 and must be 48 bytes in length.
- **RS512** - RSASSA-PKCS-v1_5 using SHA-512. The application supplied digest value must be computed using SHA-512 and must be 64 bytes in length.
- **RSNULL** - See RFC2437, a specialized use-case to enable certain TLS scenarios.

> **Note**

The DigestInfo is constructed on the server side for Sign operations that algorithms RS256, RS384 and RS512 generate.

Symmetric key algorithms (Managed HSM only)

- **AES-KW** - AES Key Wrap (RFC3394).
- **AES-GCM** - AES encryption in Galois Counter Mode (NIST SP 800-38d)
- **AES-CBC** - AES encryption in Cipher Block Chaining Mode (NIST SP 800-38a)

> **Note**
>
> Sign and verify operations algorithms must match the key type, otherwise service will return key size is incorrect error.

Key operations

Key Vault, including Managed HSM, supports the following operations on key objects:

- **Create**: Allows a client to create a key in Key Vault. The value of the key is generated by Key Vault and stored, and isn't released to the client. Asymmetric keys may be created in Key Vault.
- **Import**: Allows a client to import an existing key to Key Vault. Asymmetric keys may be imported to Key Vault using several different packaging methods within a JWK construct.
- **Update**: Allows a client with sufficient permissions to modify the metadata (key attributes) associated with a key previously stored within Key Vault.
- **Delete**: Allows a client with sufficient permissions to delete a key from Key Vault.
- **List**: Allows a client to list all keys in a given Key Vault.
- **List versions**: Allows a client to list all versions of a given key in a given Key Vault.
- **Get**: Allows a client to retrieve the public parts of a given key in a Key Vault.
- **Backup**: Exports a key in a protected form.
- **Restore**: Imports a previously backed up key.
- **Release**: It securely releases a key to authorized code running within a confidential compute environment. It requires an attestation that the Trusted Execution Environment (TEE) meets the requirements of the key's release_policy.
- **Rotate**: Rotate an existing key by generating new version of the key (Key Vault only).

For more information, see Key operations in the Key Vault REST API reference.

Once a key has been created in Key Vault, the following cryptographic operations may be performed using the key:

- **Sign and Verify**: Strictly, this operation is "sign hash" or "verify hash", as Key Vault doesn't support hashing of content as part of signature creation. Applications should hash the data to be signed locally, then request that Key Vault sign the hash. Verification of signed hashes is supported as a convenience operation for applications that may not have access to [public] key material. For best application performance, VERIFY operations should be performed locally.
- **Key Encryption / Wrapping**: A key stored in Key Vault may be used to protect another key, typically a symmetric content encryption key (CEK). When the key in Key Vault is asymmetric, key encryption is used. For example, RSA-OAEP and the WRAPKEY/UNWRAPKEY operations are equivalent to ENCRYPT/DECRYPT. When the key in Key Vault is symmetric, key wrapping is used. For example, AES-KW. The WRAPKEY operation is supported as a convenience for applications that may not have access to [public] key material. For best application performance, WRAPKEY operations should be performed locally.
- **Encrypt and Decrypt**: A key stored in Key Vault may be used to encrypt or decrypt a single block of data. The size of the block is determined by the key type and selected encryption algorithm. The Encrypt operation is provided for convenience, for applications that may not have access to [public] key material. For best application performance, ENCRYPT operations should be performed locally.

While WRAPKEY/UNWRAPKEY using asymmetric keys may seem superfluous (as the operation is equivalent to ENCRYPT/DECRYPT), the use of distinct operations is important. The distinction provides semantic and authorization separation of these operations, and consistency when other key types are supported by the service.

Key Vault doesn't support EXPORT operations. Once a key is provisioned in the system, it cannot be extracted or its key material modified. However, users of Key Vault may require their key for other use cases, such as after it has been deleted. In this case, they may use the BACKUP and RESTORE operations to export/import the key in a protected form. Keys created by the BACKUP operation are not usable outside Key Vault. Alternatively, the IMPORT operation may be used against multiple Key Vault instances.

Users may restrict any of the cryptographic operations that Key Vault supports on a per-key basis using the key_ops property of the JWK object.

For more information on JWK objects, see JSON Web Key (JWK).

Key rotation policy operations

Key vault key auto-rotation can be set by configuring key auto-rotation policy. It is only available on Key Vault resource.

- **Get Rotation Policy**: Retrieve rotation policy configuration
- **Set Rotation Policy**: Set rotation policy configuration

Key attributes

In addition to the key material, the following attributes may be specified. In a JSON Request, the attributes keyword and braces, '{' '}', are required even if there are no attributes specified.

- *enabled*: boolean, optional, default is **true**. Specifies whether the key is enabled and useable for cryptographic operations. The *enabled* attribute is used with *nbf* and *exp*. When an operation occurs between *nbf* and *exp*, it will only be permitted if *enabled* is set to **true**. Operations outside the *nbf* / *exp* window are automatically disallowed, except for decrypt, unwrap, and verify.
- *nbf*: IntDate, optional, default is now. The *nbf* (not before) attribute identifies the time before which the key MUST NOT be used for cryptographic operations, except for decrypt, unwrap, and verify. The processing of the *nbf* attribute requires that the current date/time MUST be after or equal to the not-before date/time listed in the *nbf* attribute. Key Vault MAY provide for some small leeway, normally no more than a few minutes, to account for clock skew. Its value MUST be a number containing an IntDate value.
- *exp*: IntDate, optional, default is "forever". The *exp* (expiration time) attribute identifies the expiration time on or after which the key MUST NOT be used for cryptographic operation, except for decrypt, unwrap, and verify. The processing of the *exp* attribute requires that the current date/time MUST be before the expiration date/time listed in the *exp* attribute. Key Vault MAY provide for some small leeway, typically no more than a few minutes, to account for clock skew. Its value MUST be a number containing an IntDate value.

There are more read-only attributes that are included in any response that includes key attributes:

- *created*: IntDate, optional. The *created* attribute indicates when this version of the key was created. The value is null for keys created prior to the addition of this attribute. Its value MUST be a number containing an IntDate value.
- *updated*: IntDate, optional. The *updated* attribute indicates when this version of the key was updated. The value is null for keys that were last updated prior to the

addition of this attribute. Its value MUST be a number containing an IntDate value.

For more information on IntDate and other data types, see [About keys, secrets, and certificates: Data types.

Date-time controlled operations

Not-yet-valid and expired keys, outside the *nbf* / *exp* window, will work for **decrypt**, **unwrap**, and **verify** operations (won't return 403, Forbidden). The rationale for using the not-yet-valid state is to allow a key to be tested before production use. The rationale for using the expired state is to allow recovery operations on data that was created when the key was valid. Also, you can disable access to a key using Key Vault policies, or by updating the *enabled* key attribute to **false**.

For more information on data types, see Data types.

For more information on other possible attributes, see the JSON Web Key (JWK).

Key tags

You can specify more application-specific metadata in the form of tags. Key Vault supports up to 15 tags, each of which can have a 256 character name and a 256 character value.

> **Note**
>
> Tags are readable by a caller if they have the *list* or *get* permission to that key.

Key access control

Access control for keys managed by Key Vault is provided at the level of a Key Vault that acts as the container of keys. You can control access to keys using Key Vault role-based access control (recommended) or old vault access policy permission model. Role-based permission model has three predefined roles to manage keys: 'Key Vault Crypto Officer', 'Key Vault Crypto User', 'Key Vault Service Encryption User' and can be scoped to subscription, resource group or vault level.

Vault access policy permission model permissions:

- Permissions for key management operations
  - *get*: Read the public part of a key, plus its attributes

- *list*: List the keys or versions of a key stored in a key vault
- *update*: Update the attributes for a key
- *create*: Create new keys
- *import*: Import a key to a key vault
- *delete*: Delete the key object
- *recover*: Recover a deleted key
- *backup*: Back up a key in a key vault
- *restore*: Restore a backed up key to a key vault

- Permissions for cryptographic operations
  - *decrypt*: Use the key to unprotect a sequence of bytes
  - *encrypt*: Use the key to protect an arbitrary sequence of bytes
  - *unwrapKey*: Use the key to unprotect wrapped symmetric keys
  - *wrapKey*: Use the key to protect a symmetric key
  - *verify*: Use the key to verify digests
  - *sign*: Use the key to sign digests

- Permissions for privileged operations
  - *purge*: Purge (permanently delete) a deleted key
  - *release*: Release a key to a confidential compute environment, which matches the release_policy of the key

- Permissions for rotation policy operations
  - *rotate*: Rotate an existing key by generating new version of the key (Key Vault only)
  - *get rotation policy*: Retrieve rotation policy configuration
  - *set rotation policy*: Set rotation policy configuration

For more information on working with keys, see Key operations in the Key Vault REST API reference.

Next steps

- About Key Vault
- About Managed HSM
- About secrets
- About certificates
- Key Vault REST API overview
- Authentication, requests, and responses
- Key Vault Developer's Guide

Additional resources

Documentation

About keys - Azure Key Vault

Overview of Azure Key Vault REST interface and developer details for keys.

Manage keys in a managed HSM - Azure Key Vault

Use this article to manage keys in a managed HSM

Quickstart - Provision and activate an Azure Managed HSM

Quickstart showing how to provision and activate a managed HSM using Azure CLI

Managed HSM data plane role management - Azure Key Vault

Use this article to manage role assignments for your managed HSM.

Azure Managed HSM access control

Manage access permissions for Azure Managed HSM and keys. Covers the authentication and authorization model for Managed HSM, and how to secure your HSMs.

Azure Key Vault security worlds

Azure Key Vault is a multi-tenant service. It uses a pool of HSMs in each Azure region. All regions in a geographic region share a cryptographic boundary.

encrypt - encrypt - REST API (Azure Key Vault)

Encrypts an arbitrary sequence of bytes using an encryption key that is stored in a key vault. The ENCRYPT operation encrypts an arbitrary sequence of bytes usi

Full backup/restore and selective restore for Azure Managed HSM

This document explains full backup/restore and selective restore

Show 5 more

Training

Learning paths and modules

Deploy and secure Azure Key Vault - Training

Protect your keys, certificates, and secrets in Azure Key Vault. Learn to configure key vault for the most secure deployment.

Configure cryptographic key auto-rotation in Azure Key Vault

Article • 03/14/2023 • 4 minutes to read

Overview

Automated cryptographic key rotation in Key Vault allows users to configure Key Vault to automatically generate a new key version at a specified frequency. To configure rotation you can use key rotation policy, which can be defined on each individual key.

Our recommendation is to rotate encryption keys at least every two years to meet cryptographic best practices.

For more information about how objects in Key Vault are versioned, see Key Vault objects, identifiers, and versioning.

Integration with Azure services

This feature enables end-to-end zero-touch rotation for encryption at rest for Azure services with customer-managed key (CMK) stored in Azure Key Vault. Please refer to specific Azure service documentation to see if the service covers end-to-end rotation.

For more information about data encryption in Azure, see:

- Azure Encryption at Rest
- Azure services data encryption support table

Pricing

There's an additional cost per scheduled key rotation. For more information, see Azure Key Vault pricing page

Permissions required

Key Vault key rotation feature requires key management permissions. You can assign a "Key Vault Crypto Officer" role to manage rotation policy and on-demand rotation.

For more information on how to use Key Vault RBAC permission model and assign Azure roles, see Use an Azure RBAC to control access to keys, certificates and secrets

> **Note**
>
> If you use an access policies permission model, it is required to set 'Rotate', 'Set Rotation Policy', and 'Get Rotation Policy' key permissions to manage rotation policy on keys.

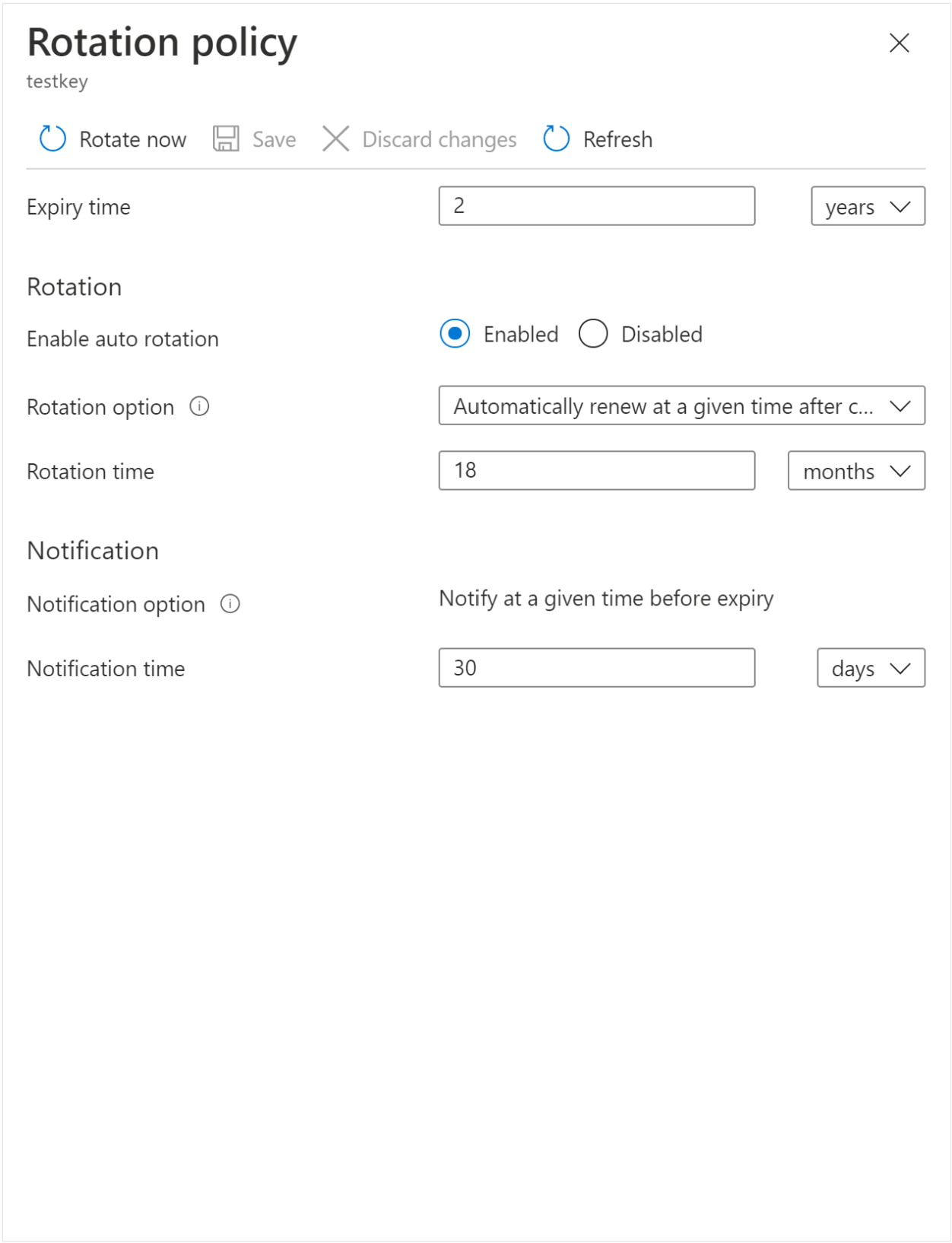
Key rotation policy

The key rotation policy allows users to configure rotation and Event Grid notifications near expiry notification.

Key rotation policy settings:

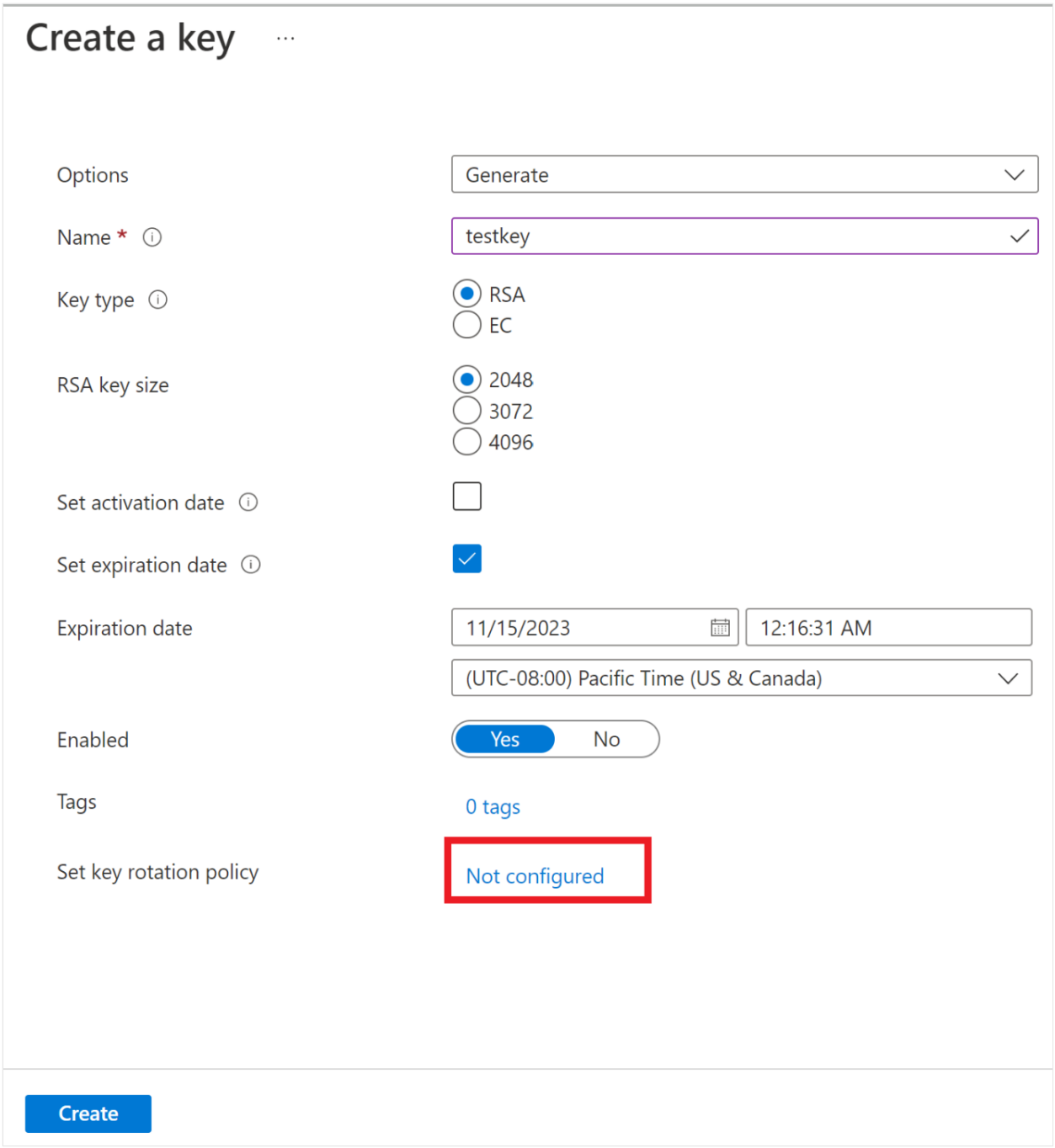
- Expiry time: key expiration interval. It's used to set expiration date on newly rotated key. It doesn't affect a current key.
- Enabled/disabled: flag to enable or disable rotation for the key
- Rotation types:
  - Automatically renew at a given time after creation (default)
  - Automatically renew at a given time before expiry. It requires 'Expiry Time' set on rotation policy and 'Expiration Date' set on the key.
- Rotation time: key rotation interval, the minimum value is seven days from creation and seven days from expiration time
- Notification time: key near expiry event interval for Event Grid notification. It requires 'Expiry Time' set on rotation policy and 'Expiration Date' set on the key.

> **Important**
>
> Key rotation generates a new key version of an existing key with new key material. Target services should use versionless key uri to automatically refresh to latest version of the key. Ensure that your data encryption solution stores versioned key uri with data to point to the same key material for decrypt/unwrap as was used for encrypt/wrap operations to avoid disruption to your services. All Azure services are currently following that pattern for data encryption.

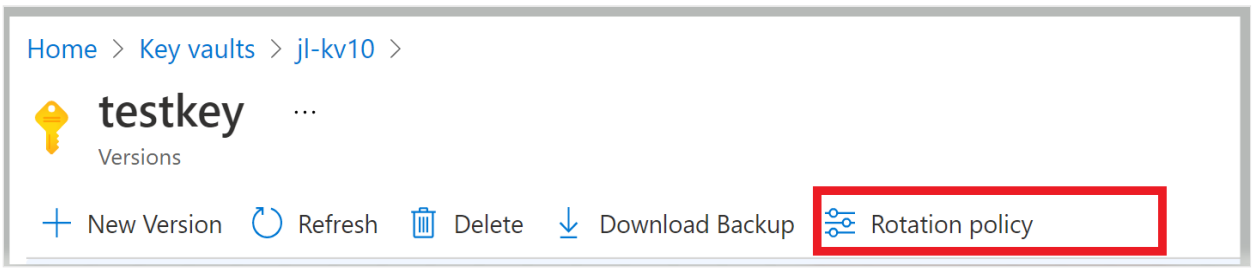


Configure key rotation policy

Configure key rotation policy during key creation.



Configure rotation policy on existing keys.



Azure CLI

Save key rotation policy to a file. Key rotation policy example:

JSON

```
{
  "lifetimeActions": [
    {
      "trigger": {
        "timeAfterCreate": "P18M",
        "timeBeforeExpiry": null
      },
      "action": {
        "type": "Rotate"
      }
    },
    {
      "trigger": {
        "timeBeforeExpiry": "P30D"
      },
      "action": {
        "type": "Notify"
      }
    }
  ],
  "attributes": {
    "expiryTime": "P2Y"
  }
}
```

Set rotation policy on a key passing previously saved file using Azure CLI az keyvault key rotation-policy update command.

Azure CLI

```
az keyvault key rotation-policy update --vault-name <vault-name> --name
<key-name> --value </path/to/policy.json>
```

Azure PowerShell

Set rotation policy using Azure Powershell Set-AzKeyVaultKeyRotationPolicy cmdlet.

PowerShell

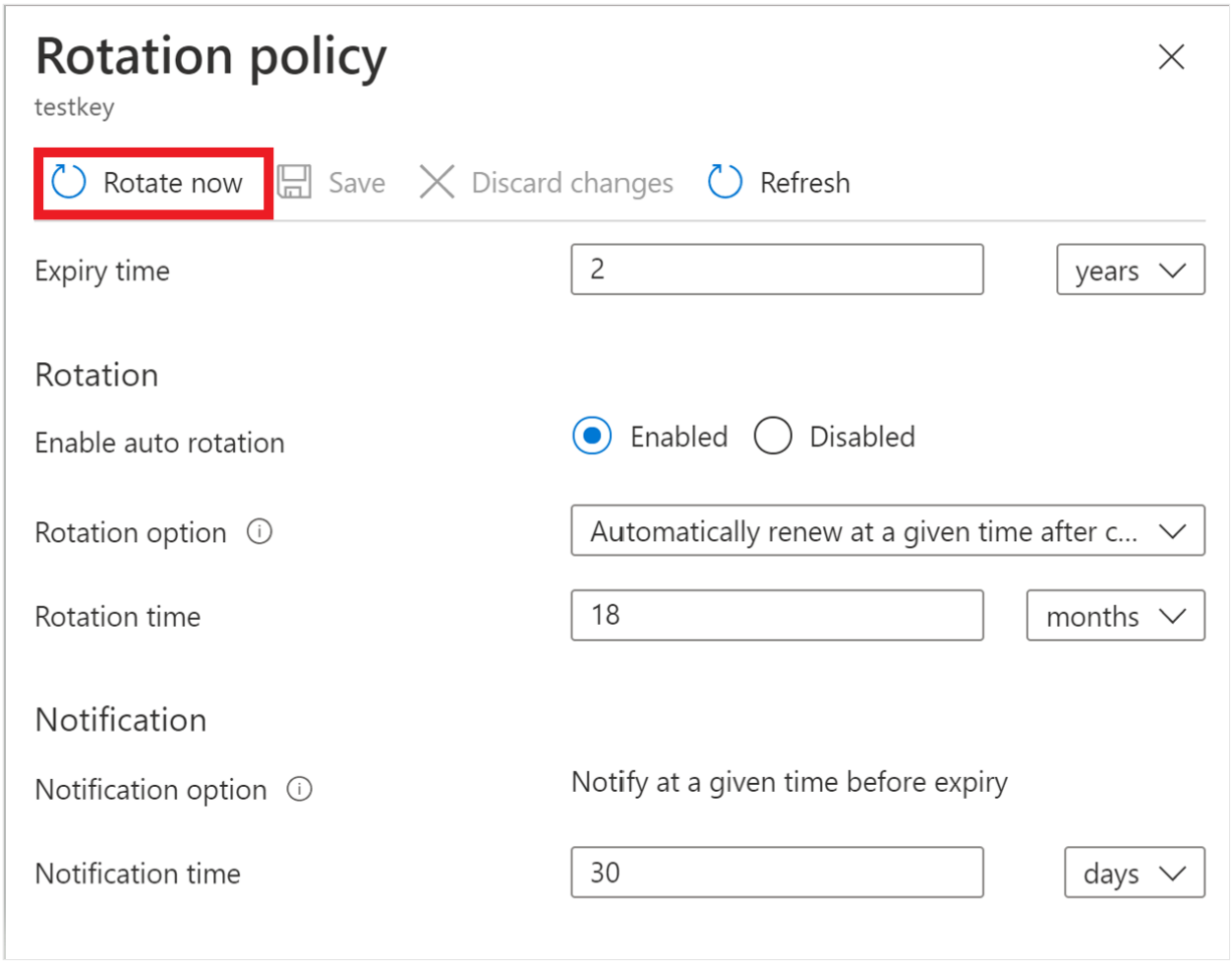
```
Set-AzKeyVaultKeyRotationPolicy -VaultName <vault-name> -KeyName <key-name>
-ExpiresIn (New-TimeSpan -Days 720) -KeyRotationLifetimeAction
@{Action="Rotate";TimeAfterCreate= (New-TimeSpan -Days 540)}
```

Rotation on demand

Key rotation can be invoked manually.

Portal

Click 'Rotate Now' to invoke rotation.



Azure CLI

Use Azure CLI az keyvault key rotate command to rotate key.

Azure CLI

```
az keyvault key rotate --vault-name <vault-name> --name <key-name>
```

Azure PowerShell

Use Azure PowerShell Invoke-AzKeyVaultKeyRotation cmdlet.

PowerShell

```
Invoke-AzKeyVaultKeyRotation -VaultName <vault-name> -Name <key-name>
```

Configure key near expiry notification

Configuration of expiry notification for Event Grid key near expiry event. In case when automated rotation cannot be used, like when a key is imported from local HSM, you can configure near expiry notification as a reminder for manual rotation or as a trigger to custom automated rotation through integration with Event Grid. You can configure notification with days, months and years before expiry to trigger near expiry event.

For more information about Event Grid notifications in Key Vault, see Azure Key Vault as Event Grid source

Configure key rotation with ARM template

Key rotation policy can also be configured using ARM templates.

> **Note**
>
> It requires 'Key Vault Contributor' role on Key Vault configured with Azure RBAC to deploy key through management plane.

JSON

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "vaultName": {
            "type": "String",
            "metadata": {
                "description": "The name of the key vault to be created."
            }
        },
        "keyName": {
            "type": "String",
            "metadata": {
                "description": "The name of the key to be created."
            }
```

```
        },
        "rotatationTimeAfterCreate": {
            "defaultValue": "P18M",
            "type": "String",
            "metadata": {
                "description": "Time duration to trigger key rotation. i.e.
P30D, P1M, P2Y"
            }
        },
        "expiryTime": {
            "defaultValue": "P2Y",
            "type": "String",
            "metadata": {
                "description": "The expiry time for new key version. i.e.
P90D, P2M, P3Y"
            }
        },
        "notifyTime": {
            "defaultValue": "P30D",
            "type": "String",
            "metadata": {
                "description": "Near expiry Event Grid notification. i.e.
P30D"
            }
        }

    },
    "resources": [
        {
            "type": "Microsoft.KeyVault/vaults/keys",
            "apiVersion": "2021-06-01-preview",
            "name": "[concat(parameters('vaultName'), '/',
parameters('keyName'))]",
            "location": "[resourceGroup().location]",
            "properties": {
                "vaultName": "[parameters('vaultName')]",
                "kty": "RSA",
                "rotationPolicy": {
                    "lifetimeActions": [
                        {
                            "trigger": {
                                "timeAfterCreate": "
[parameters('rotatationTimeAfterCreate')]",
                                "timeBeforeExpiry": ""
                            },
                            "action": {
                                "type": "Rotate"
                            }
                        },
                        {
                            "trigger": {
                                "timeBeforeExpiry": "
[parameters('notifyTime')]"
                            },
                            "action": {
```

```
                            "type": "Notify"
                        }
                    }

                ],
                "attributes": {
                    "expiryTime": "[parameters('expiryTime')]"
                }
            }
        }
    }
]
}
```

Resources

- Monitoring Key Vault with Azure Event Grid
- Use an Azure RBAC to control access to keys, certificates and secrets
- Azure Data Encryption At Rest
- Azure Storage Encryption
- Azure Disk Encryption
- Automatic key rotation for transparent data encryption

Azure Key Vault secure key release policy grammar

Article • 03/21/2022 • 4 minutes to read

This article documents a simplified EBNF grammar for secure key release policy, which itself is modeled on Azure Policy. For a complete example of a secure key release policy, see the confidential VM key release policy.

JSON

```
(* string and number from JSON *)
value =
  string |
  number |
  "true" |
  "false";

(* The operators supported for claim value comparison *)
operator =
  "equals:" |
  "notEquals:" |
  "less:" |
  "lessOrEquals:" |
  "greater:" |
  "greaterOrEquals:" |
  "exists:";

(* A JSON condition that evaluates the value of a claim *)
claim_condition =
  "{" "claim:", string "," operator, ":", value "}";

(* A JSON condition requiring any of the listed conditions to be true *)
anyof_condition =
  "{" "anyof:", condition_array "}";

(* A JSON condition requiring all of the listed conditions to be true *)
allof_condition =
  "{" "allof:", condition_array "}";

(* A condition is any of the allowed condition types *)
condition =
  claim_condition |
  anyof_condition |
  allof_condition;

(* A list of conditions, one is required *)
condition_list =
  condition { "," condition };

(* An JSON array of conditions *)
```

```
condition_array =
  "[" condition_list "]";

(* A JSON authority with its conditions *)
authority =
  "{" "authority:", string "," ( anyof_condition | allof_condition );

(* A list of authorities, one is required *)
authority_list =
  authority { "," authority_list };

(* A policy is an anyOf selector of authorities *)
policy =
  "{" "version: \"1.0.0\"", "anyOf:", "[" authority_list "]" "}";
```

Claim condition

A Claim Condition is a JSON object that identifies a claim name, a condition for matching, and a value, for example:

JSON

```
{
  "claim": "<claim name>",
  "equals": <value to match>
}
```

In the first iteration, the only allowed condition is "equals" but future iterations may allow for other operators similar to Azure Policy (see the section on Conditions). If a specified claim isn't present, its condition is considered to haven't been met.

Claim names allow "dot notation" to enable JSON object navigation, for example:

JSON

```
{
  "claim": "object.object.claim",
  "equals": <value to match>
}
```

Array specifications aren't presently supported. Per the grammar, objects aren't allowed as values for matching.

AnyOf, AllOf conditions

AnOf and AllOf condition objects allow for the modeling of OR and AND. For AnyOf, if any of the conditions provided are true, the condition is met. For AllOf, all of the conditions must be true.

Examples are shown below. In the first, allOf requires all conditions to be met:

JSON

```
{
  "allOf":
  [
    {
      "claim": "<claim_1>",
      "equals": <value_1>
    },
    {
      "claim": "<claim_2>",
      "equals": <value_2>
    }
  ]
}
```

Meaning (claim_1 == value_1) && (claim_2 == value_2).

In this example, anyOf requires that any condition match:

JSON

```
{
  "anyOf":
  [
    {
      "claim": "<claim_1>",
      "equals": <value_1>
    },
    {
      "claim": "<claim_2>",
      "equals": <value_2>
    }
  ]
}
```

Meaning (claim_1 == value_2) || (claim_2 == value_2)

The anyOf and allOf condition objects may be nested:

JSON

```
  "allOf":
  [
```

```
    {
      "claim": "<claim_1>",
      "equals": <value_1>
    },
    {
      "anyOf":
      [
        {
          "claim": "<claim_2>",
          "equals": <value_2>
        },
        {
          "claim": "<claim_3>",
          "equals": <value_3>
        }
      ]
    }
  ]
```

Or:

JSON

```
{
  "allOf":
  [
    {
      "claim": "<claim_1>",
      "equals": <value_1>
    },
    {
      "anyOf":
      [
        {
          "claim": "<claim_2>",
          "equals": <value_2>
        },
        {
          "allOf":
          [
            {
              "claim": "<claim_3>",
              "equals": <value_3>
            },
            {
              "claim": "<claim_4>",
              "equals": <value_4>
            }
          ]
        }
      ]
    }
```

```
  ]
}
```

Key release authority

Conditions are collected into Authority statements and combined:

JSON

```
{
  "authority": "<issuer>",
  "allOf":
  [
    {
      "claim": "<claim_1>",
      "equals": <value_1>
    }
  ]
}
```

Where:

- **authority**: An identifier for the authority making the claims. This identifier functions in the same fashion as the iss claim in a JSON Web Token. It indirectly references a key that signs the Environment Assertion.
- **allOf**: One or more claim conditions that identify claims and values that must be satisfied in the environment assertion for the release policy to succeed. anyOf is also allowed. However, both aren't allowed together.

Key Release Policy

Release policy is an anyOf condition containing an array of key authorities:

JSON

```
{
  "anyOf":
  [
    {
      "authority": "my.attestation.com",
      "allOf":
      [
        {
          "claim": "mr-signer",
          "equals": "0123456789"
        }
```

```
      ]
    }
  ]
}
```

Encoding key release policy

Since key release policy is a JSON document, it's encoded when carried in requests and response to AKV to avoid the need to describe the complete language in Swagger definitions.

The encoding is as follows:

JSON

```
{
  "contentType": "application/json; charset=utf-8",
  "data": "<BASE64URL(JSON serialization of policy)>"
}
```

Environment Assertion

An Environment Assertion is a signed assertion, in JSON Web Token form, from a trusted authority. An Environment Asserting contains at least a key encryption key and one or more claims about the target environment (for example, TEE type, publisher, version) that are matched against the Key Release Policy. The key encryption key is a public RSA key owned and protected by the target execution environment that is used for key export. It must appear in the TEE keys claim (x-ms-runtime/keys). This claim is a JSON object representing a JSON Web Key Set. Within the JWKS, one of the keys must meet the requirements for use as an encryption key (key_use is "enc", or key_ops contains "encrypt"). The first suitable key is chosen.

Additional resources

🕮 Documentation

Managed HSM local RBAC built-in roles - Azure Key Vault

An overview of Managed HSM built-in roles that can be assigned to users, service principals, groups, and managed identities

Create and retrieve attributes of a managed key in Azure Key Vault – Azure

PowerShell

Quickstart showing how to set and retrieve a managed key from Azure Key Vault using Azure PowerShell

az keyvault security-domain

Managed Hsms - REST API (Azure Key Vault)

Learn more about [Key Vault Managed Hsms Operations]. How to [Check Mhsm Name Availability,Create Or Update,Delete,Get,Get Deleted,List By Resource Group,List B

Quickstart - Provision and activate an Azure Managed HSM

Quickstart showing how to provision and activate a managed HSM using Azure CLI

Manage keys in a managed HSM - Azure Key Vault

Use this article to manage keys in a managed HSM

Azure Key Vault Managed HSM recovery overview

Managed HSM recovery features are designed to prevent the accidental or malicious deletion of your HSM resource and keys.

Managed HSM data plane role management - Azure Key Vault

Use this article to manage role assignments for your managed HSM.

Show 5 more

Key management in Azure

Article • 02/28/2023 • 5 minutes to read

> **Note**
>
> **Zero Trust** is a security strategy comprising three principles: "Verify explicitly", "Use least privilege access", and "Assume breach". Data protection, including key management, supports the "use least privilege access" principle. For more information, see **What is Zero Trust?**

In Azure, encryption keys can be either platform managed or customer managed.

Platform-managed keys (PMKs) are encryption keys that are generated, stored, and managed entirely by Azure. Customers do not interact with PMKs. The keys used for Azure Data Encryption-at-Rest, for instance, are PMKs by default.

Customer-managed keys (CMK), on the other hand, are those that can be read, created, deleted, updated, and/or administered by one or more customers. Keys stored in a customer-owned key vault or hardware security module (HSM) are CMKs. Bring Your Own Key (BYOK) is a CMK scenario in which a customer imports (brings) keys from an outside storage location into an Azure key management service (see the Azure Key Vault: Bring your own key specification).

A specific kind of customer-managed key is the "key encryption key" (KEK). A KEK is a primary key that controls access to one or more encryption keys that are themselves encrypted.

Customer-managed keys can be stored on-premises or, more commonly, in a cloud key management service.

Azure key management services

Azure offers several options for storing and managing your keys in the cloud, including Azure Key Vault, Azure Managed HSM, Dedicated HSM, and Payments HSM. These options differ in terms of their FIPS compliance level, management overhead, and intended applications.

Azure Key Vault (Standard Tier): A FIPS 140-2 Level 1 validated multi-tenant cloud key management service that can also be used to store secrets and certificates. Keys stored in Azure Key Vault are software-protected and can be used for encryption-at-rest and

custom applications. Key Vault provides a modern API and the widest breadth of regional deployments and integrations with Azure Services. For more information, see About Azure Key Vault.

Azure Key Vault (Premium Tier): A FIPS 140-2 Level 2 validated multi-tenant HSM offering that can be used to store keys in a secure hardware boundary. Microsoft manages and operates the underlying HSM, and keys stored in Azure Key Vault Premium can be used for encryption-at-rest and custom applications. Key Vault Premium also provides a modern API and the widest breadth of regional deployments and integrations with Azure Services. For more information, see About Azure Key Vault.

Azure Managed HSM: A FIPS 140-2 Level 3 validated single-tenant HSM offering that gives customers full control of an HSM for encryption-at-rest, Keyless SSL, and custom applications. Customers receive a pool of three HSM partitions—together acting as one logical, highly available HSM appliance--fronted by a service that exposes crypto functionality through the Key Vault API. Microsoft handles the provisioning, patching, maintenance, and hardware failover of the HSMs, but doesn't have access to the keys themselves, because the service executes within Azure's Confidential Compute Infrastructure. Managed HSM is integrated with the Azure SQL, Azure Storage, and Azure Information Protection PaaS services and offers support for Keyless TLS with F5 and Nginx. For more information, see What is Azure Key Vault Managed HSM?

Azure Dedicated HSM: A FIPS 140-2 Level 3 validated bare metal HSM offering, that lets customers lease a general-purpose HSM appliance that resides in Microsoft datacenters. The customer has complete and total ownership over the HSM device and is responsible for patching and updating the firmware when required. Microsoft has no permissions on the device or access to the key material, and Dedicated HSM is not integrated with any Azure PaaS offerings. Customers can interact with the HSM using the PKCS#11, JCE/JCA, and KSP/CNG APIs. This offering is most useful for legacy lift-and-shift workloads, PKI, SSL Offloading and Keyless TLS (supported integrations include F5, Nginx, Apache, Palo Alto, IBM GW and more), OpenSSL applications, Oracle TDE, and Azure SQL TDE IaaS. For more information, see What is Azure Key Vault Managed HSM?

Azure Payments HSM: A FIPS 140-2 Level 3, PCI HSM v3, validated bare metal offering that lets customers lease a payment HSM appliance in Microsoft datacenters for payments operations, including payment processing, payment credential issuing, securing keys and authentication data, and sensitive data protection. The service is PCI DSS and PCI 3DS compliant. Azure Payment HSM offers single-tenant HSMs for customers to have complete administrative control and exclusive access to the HSM. Once the HSM is allocated to a customer, Microsoft has no access to customer data. Likewise, when the HSM is no longer required, customer data is zeroized and erased as

soon as the HSM is released, to ensure complete privacy and security is maintained. For more information, see About Azure Payment HSM.

Pricing

The Azure Key Vault Standard and Premium tiers are billed on a transactional basis, with an additional monthly per-key charge for premium hardware-backed keys. Managed HSM, Dedicated HSM, and Payments HSM don't charge on a transactional basis; instead they are always-in-use devices that are billed at a fixed hourly rate. For detailed pricing information, see Key Vault pricing, Dedicated HSM pricing, and Payment HSM pricing.

Service Limits

Managed HSM, Dedicated HSM, and Payments HSM offer dedicated capacity. Key Vault Standard and Premium are multi-tenant offerings and have throttling limits. For service limits, see Key Vault service limits.

Encryption-At-Rest

Azure Key Vault and Azure Key Vault Managed HSM have integrations with Azure Services and Microsoft 365 for Customer Managed Keys, meaning customers may use their own keys in Azure Key Vault and Azure Key Managed HSM for encryption-at-rest of data stored in these services. Dedicated HSM and Payments HSM are Infrastructure-as-Service offerings and do not offer integrations with Azure Services. For an overview of encryption-at-rest with Azure Key Vault and Managed HSM, see Azure Data Encryption-at-Rest.

APIs

Dedicated HSM and Payments HSM support the PKCS#11, JCE/JCA, and KSP/CNG APIs, but Azure Key Vault and Managed HSM do not. Azure Key Vault and Managed HSM use the Azure Key Vault REST API and offer SDK support. For more information on the Azure Key Vault API, see Azure Key Vault REST API Reference.

What's next

- Azure Key Vault
- Azure Managed HSM
- Azure Dedicated HSM

- Azure Payment HSM
- What is Zero Trust?

Additional resources

Documentation

Best practices using Azure Key Vault Managed HSM

This document explains some of the best practices to use Key Vault

Azure Managed HSM Overview - Azure Managed HSM

Azure Managed HSM is a cloud service that safeguards your cryptographic keys for cloud applications.

About Azure Managed HSM security domain

Overview of the Managed HSM Security Domain, a set of core credentials needed to recover a Managed HSM

Azure Key Vault security worlds

Azure Key Vault is a multi-tenant service. It uses a pool of HSMs in each Azure region. All regions in a geographic region share a cryptographic boundary.

Full backup/restore and selective restore for Azure Managed HSM

This document explains full backup/restore and selective restore

Azure Managed HSM access control

Manage access permissions for Azure Managed HSM and keys. Covers the authentication and authorization model for Managed HSM, and how to secure your HSMs.

Azure security baseline for Key Vault

The Key Vault security baseline provides procedural guidance and resources for implementing the security recommendations specified in the Azure Security Benchmark.

Show 4 more

Multitenancy and Azure Key Vault

Article • 09/01/2022 • 6 minutes to read

Azure Key Vault is used to manage secure data for your solution, including secrets, encryption keys, and certificates. In this article, we describe some of the features of Azure Key Vault that are useful for multitenant solutions. We then provide links to the guidance that can help you, when you're planning how you're going to use Key Vault.

Isolation models

When working with a multitenant system using Key Vault, you need to make a decision about the level of isolation that you want to use. The choice of isolation models you use depends on the following factors:

- How many tenants do you plan to have?
- Do you share your application tier between multiple tenants, do you deploy single-tenant application instances, or do you deploy separate deployment stamps for each tenant?
- Do your tenants need to manage their own encryption keys?
- Do your tenants have compliance requirements that require their secrets are stored separately from other tenants' secrets?

The following table summarizes the differences between the main tenancy models for Key Vault:

Consideration	Vault per tenant, in the provider's subscription	Vault per tenant, in the tenant's subscription	Shared vault
Data isolation	High	Very high	Low
Performance isolation	Medium. High throughput might be limited, even with many vaults	High	Low
Deployment complexity	Low-medium, depending on the number of tenants	High. The tenant must correctly grant access to the provider	Low
Operational complexity	High	Low for the provider, higher for the tenant	Lowest
Example scenario	Individual application instances per tenant	Customer-managed encryption keys	Large multitenant solution with a shared application tier

Vault per tenant, in the provider's subscription

You might consider deploying a vault for each of your tenants within your (the service provider's) Azure subscription. This approach provides you with strong data isolation between each tenant's data, but it requires that you deploy and manage an increasing number of vaults, as you increase the number of tenants.

This approach makes sense when you have separate application deployments for each tenant. If you have a shared application tier, it's unlikely that using separate vaults will give you much data isolation benefit, because all of the vaults need to trust the same application tier.

There's no limit to the number of vaults you can deploy into an Azure subscription. However, you should consider the following limits:

- There are subscription-wide limits on the number of requests in a time period. These limits apply regardless of the number of vaults in the subscription. So, it's important to follow our throttling guidance, even when you have tenant-specific vaults.
- There's a limit to the number of Azure role assignments that you can create within a subscription. When you deploy and configure large numbers of vaults in a subscription, you might approach these limits.

Vault per tenant, in the tenant's subscription

In some situations, your tenants might create vaults in their own Azure subscriptions, and they might want to grant your application access to work with secrets, certificates, or keys. This approach is appropriate when you allow *customer-managed keys* (CMKs) for encryption within your solution.

In order to access the data in your tenant's vault, the tenant must provide your application with access to their vault. This process requires that your application authenticates through their Azure AD instance. One approach is to publish a multitenant Azure AD application. Your tenants must perform a one-time consent process. They first register the multitenant Azure AD application in their own Azure AD tenant. Then, they grant your multitenant Azure AD application the appropriate level of access to their vault. They also need to provide you with the full resource ID of the vault that they've created. Then, your application code can use a service principal that's associated with the multitenant Azure AD application in your own Azure AD, to access each tenant's vault.

Alternatively, you might ask each tenant to create a service principal for your service to use, and to provide you with its credentials. However, this approach requires that you securely store and manage credentials for each tenant, which is a potential security liability.

If your tenants configure network access controls on their vaults, make sure you'll be able to access the vaults.

Shared vaults

You might choose to share tenants' secrets within a single vault. The vault is deployed in your (the solution provider's) Azure subscription, and you're responsible for managing it. This approach is simplest, but it provides the least data isolation and performance isolation.

You might also choose to deploy multiple shared vaults. For example, if you follow the Deployment Stamps pattern, it's likely you'll deploy a shared vault within each stamp. Similarly, if you deploy a multi-region solution, you should deploy vaults into each region for the following reasons:

- To avoid cross-region traffic latency when working with the data in your vault.
- To support data residency requirements.
- To enable the use of regional vaults within other services that require same-region deployments.

When you work with a shared vault, it's important to consider the number of operations you perform against the vault. Operations include reading secrets and performing encryption or decryption operations. Key Vault imposes limits on the number of requests that can be made against a single vault, and across all of the vaults within an Azure subscription. Ensure that you follow the throttling guidance. It's important to follow the recommended practices, including securely caching the secrets that you retrieve and using envelope encryption to avoid sending every encryption operation to Key Vault. When you follow these best practices, you can run high-scale solutions against a single vault.

If you need to store tenant-specific secrets, keys, or certificates, consider using a naming convention like a naming prefix. For example, you might prepend the tenant ID to the name of each secret. Then, your application code can easily load the value of a specific secret for a specific tenant.

Features of Azure Key Vault that support multitenancy

Tags

Key Vault supports tagging secrets, certificates, and keys with custom metadata, so you can use a tag to track the tenant ID for each tenant-specific secret. However, Key Vault doesn't support querying by tags, so this feature is best suited for management purposes, rather than for use within your application logic.

More information:

- Secret tags
- Certificate tags
- Key tags

Azure Policy support

If you decide to deploy a large number of vaults, it's important to ensure that they follow a consistent standard for network access configuration, logging, and access control. Consider using Azure Policy to verify the vaults have been configured according to your requirements.

More information:

- Integrate Azure Key Vault with Azure Policy
- Azure Policy built-in definitions for Key Vault

Managed HSM and Dedicated HSM

If you need to perform a large number of operations per second, and the Key Vault operation limits are insufficient, consider using either Managed HSM or Dedicated HSM. Both products provide you with a reserved amount of capacity, but they're usually more costly than Key Vault. Additionally, be aware of the limits on the number of instances of these services that you can deploy into each region.

More information:

- How do I decide whether to use Azure Key Vault or Azure Dedicated HSM?
- Is Azure Dedicated HSM right for you?

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- John Downs | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- Jack Lichwa | Principal Product Manager, Azure Key Vault
- Arsen Vladimirskiy | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review deployment and configuration approaches for multitenancy.

Az.KeyVault

Reference

This topic displays help topics for the Azure Key Vault Cmdlets.

Key Vault

Add-AzKeyVaultCertificate	Adds a certificate to a key vault.
Add-AzKeyVaultCertificateContact	Adds a contact for certificate notifications.
Add-AzKeyVaultKey	Creates a key in a key vault or imports a key into a key vault.
Add-AzKeyVaultManagedStorageAccount	Adds an existing Azure Storage Account to the specified key vault for its keys to be managed by the Key Vault service.
Add-AzKeyVaultNetworkRule	Adds a rule meant to restrict access to a key vault based on the client's internet address.
Backup-AzKeyVault	Fully backup a managed HSM.
Backup-AzKeyVaultCertificate	Backs up a certificate in a key vault.
Backup-AzKeyVaultKey	Backs up a key in a key vault.
Backup-AzKeyVaultManagedStorageAccount	Backs up a KeyVault-managed storage account.
Backup-AzKeyVaultSecret	Backs up a secret in a key vault.
Export-AzKeyVaultSecurityDomain	Exports the security domain data of a managed HSM.
Get-AzKeyVault	Gets key vaults.
Get-AzKeyVaultCertificate	Gets a certificate from a key vault.
Get-AzKeyVaultCertificateContact	Gets contacts that are registered for certificate notifications for a key vault.
Get-AzKeyVaultCertificateIssuer	Gets a certificate issuer for a key vault.
Get-AzKeyVaultCertificateOperation	Gets the status of a certificate operation.
Get-AzKeyVaultCertificatePolicy	Gets the policy for a certificate in a key vault.
Get-AzKeyVaultKey	Gets Key Vault keys. Please notes that

	detailed information about a key, like key type or key size, only available when querying a specific key version.
Get-AzKeyVaultKeyRotationPolicy	Gets the key rotation policy for the specified key in Key Vault.
Get-AzKeyVaultManagedHsm	Get managed HSMs.
Get-AzKeyVaultManagedStorageAccount	Gets Key Vault managed Azure Storage Accounts.
Get-AzKeyVaultManagedStorageSasDefinition	Gets Key Vault managed Storage SAS Definitions.
Get-AzKeyVaultRandomNumber	Get the requested number of bytes containing random values from a managed HSM.
Get-AzKeyVaultRoleAssignment	Get or list role assignments of a managed HSM. Use respective parameters to list assignments to a specific user or a role definition.
Get-AzKeyVaultRoleDefinition	List role definitions of a given managed HSM at a given scope.
Get-AzKeyVaultSecret	Gets the secrets in a key vault.
Import-AzKeyVaultCertificate	Imports a certificate to a key vault.
Import-AzKeyVaultSecurityDomain	Imports previously exported security domain data to a managed HSM.
Invoke-AzKeyVaultKeyOperation	Performs operation like "Encrypt", "Decrypt", "Wrap" or "Unwrap" using a specified key stored in a key vault or managed hsm.
Invoke-AzKeyVaultKeyRotation	Creates a new key version in Key Vault, stores it, then returns the new key.
New-AzKeyVault	Creates a key vault.
New-AzKeyVaultCertificateAdministratorDetail	Creates an in-memory certificate administrator details object.
New-AzKeyVaultCertificateOrganizationDetail	Creates an in-memory certificate organization details object.
New-AzKeyVaultCertificatePolicy	Creates an in-memory certificate policy object.
New-AzKeyVaultManagedHsm	Creates a managed HSM.

New-AzKeyVaultNetworkRuleSetObject	Create an object representing the network rule settings.
New-AzKeyVaultRoleAssignment	Assigns the specified RBAC role to the specified principal, at the specified scope.
New-AzKeyVaultRoleDefinition	Creates a custom role definition on an HSM.
Remove-AzKeyVault	Deletes a key vault.
Remove-AzKeyVaultAccessPolicy	Removes all permissions for a user or application from a key vault.
Remove-AzKeyVaultCertificate	Removes a certificate from a key vault.
Remove-AzKeyVaultCertificateContact	Deletes a contact that is registered for certificate notifications from a key vault.
Remove-AzKeyVaultCertificateIssuer	Deletes a certificate issuer from a key vault.
Remove-AzKeyVaultCertificateOperation	Deletes a certificate operation from a key vault.
Remove-AzKeyVaultKey	Deletes a key in a key vault.
Remove-AzKeyVaultManagedHsm	Deletes/Purges a managed HSM.
Remove-AzKeyVaultManagedStorageAccount	Removes a Key Vault managed Azure Storage Account and all associated SAS definitions.
Remove-AzKeyVaultManagedStorageSasDefinition	Removes a Key Vault managed Azure Storage SAS definitions.
Remove-AzKeyVaultNetworkRule	Removes a network rule from a key vault.
Remove-AzKeyVaultRoleAssignment	Removes a role assignment to the specified principal who is assigned to a particular role at a particular scope.
Remove-AzKeyVaultRoleDefinition	Removes a custom role definition from an HSM.
Remove-AzKeyVaultSecret	Deletes a secret in a key vault.
Restore-AzKeyVault	Fully restores a managed HSM from backup.
Restore-AzKeyVaultCertificate	Restores a certificate in a key vault from a backup file.
Restore-AzKeyVaultKey	Creates a key in a key vault from a backed-up key.
Restore-AzKeyVaultManagedStorageAccount	Restores a managed storage account in a key

	vault from a backup file.
Restore-AzKeyVaultSecret	Creates a secret in a key vault from a backed-up secret.
Set-AzKeyVaultAccessPolicy	Grants or modifies existing permissions for a user, application, or security group to perform operations with a key vault.
Set-AzKeyVaultCertificateIssuer	Sets a certificate issuer in a key vault.
Set-AzKeyVaultCertificatePolicy	Creates or updates the policy for a certificate in a key vault.
Set-AzKeyVaultKeyRotationPolicy	Sets the key rotation policy for the specified key in Key Vault.
Set-AzKeyVaultManagedStorageSasDefinition	Sets a Shared Access Signature (SAS) definition with Key Vault for a given Key Vault managed Azure Storage Account.
Set-AzKeyVaultSecret	Creates or updates a secret in a key vault.
Stop-AzKeyVaultCertificateOperation	Cancels a certificate operation in key vault.
Undo-AzKeyVaultCertificateRemoval	Recovers a deleted certificate in a key vault into an active state.
Undo-AzKeyVaultKeyRemoval	Recovers a deleted key in a key vault into an active state.
Undo-AzKeyVaultManagedHsmRemoval	Recover a managed HSM.
Undo-AzKeyVaultManagedStorageAccountRemoval	Recovers a previously deleted KeyVault-managed storage account.
Undo-AzKeyVaultManagedStorageSasDefinitionRemoval	Recovers a previously deleted KeyVault-managed storage SAS definition.
Undo-AzKeyVaultRemoval	Recovers a deleted key vault into an active state.
Undo-AzKeyVaultSecretRemoval	Recovers a deleted secret in a key vault into an active state.
Update-AzKeyVault	Update the state of an Azure key vault.
Update-AzKeyVaultCertificate	Modifies editable attributes of a certificate.
Update-AzKeyVaultKey	Updates the attributes of a key in a key vault.
Update-AzKeyVaultManagedHsm	Update the state of an Azure managed HSM.

Update-AzKeyVaultManagedStorageAccount	Update editable attributes of a Key Vault managed Azure Storage Account.
Update-AzKeyVaultManagedStorageAccountKey	Regenerates the specified key of Key Vault managed Azure Storage Account.
Update-AzKeyVaultNetworkRuleSet	Updates the network rule set on a key vault.
Update-AzKeyVaultSecret	Updates attributes of a secret in a key vault.

az keyvault

Reference

Manage KeyVault keys, secrets, and certificates.

Commands

az keyvault backup	Manage full HSM backup.
az keyvault backup start	Begin a full backup of the HSM.
az keyvault certificate	Manage certificates.
az keyvault certificate backup	Backs up the specified certificate.
az keyvault certificate contact	Manage contacts for certificate management.
az keyvault certificate contact add	Add a contact to the specified vault to receive notifications of certificate operations.
az keyvault certificate contact delete	Remove a certificate contact from the specified vault.
az keyvault certificate contact list	Lists the certificate contacts for a specified key vault.
az keyvault certificate create	Create a Key Vault certificate.
az keyvault certificate delete	Deletes a certificate from a specified key vault.
az keyvault certificate download	Download the public portion of a Key Vault certificate.
az keyvault certificate get-default-policy	Get the default policy for self-signed certificates.
az keyvault certificate import	Import a certificate into KeyVault.
az keyvault certificate issuer	Manage certificate issuer information.
az keyvault certificate issuer admin	Manage admin information for certificate issuers.
az keyvault certificate issuer admin add	Add admin details for a specified certificate issuer.
az keyvault certificate issuer admin delete	Remove admin details for the specified certificate issuer.

az keyvault certificate issuer admin list	List admins for a specified certificate issuer.
az keyvault certificate issuer create	Create a certificate issuer record.
az keyvault certificate issuer delete	Deletes the specified certificate issuer.
az keyvault certificate issuer list	List certificate issuers for a specified key vault.
az keyvault certificate issuer show	Lists the specified certificate issuer.
az keyvault certificate issuer update	Update a certificate issuer record.
az keyvault certificate list	List certificates in a specified key vault.
az keyvault certificate list-deleted	Lists the deleted certificates in the specified vault currently available for recovery.
az keyvault certificate list-versions	List the versions of a certificate.
az keyvault certificate pending	Manage pending certificate creation operations.
az keyvault certificate pending delete	Deletes the creation operation for a specific certificate.
az keyvault certificate pending merge	Merges a certificate or a certificate chain with a key pair existing on the server.
az keyvault certificate pending show	Gets the creation operation of a certificate.
az keyvault certificate purge	Permanently deletes the specified deleted certificate.
az keyvault certificate recover	Recovers the deleted certificate back to its current version under /certificates.
az keyvault certificate restore	Restores a backed up certificate to a vault.
az keyvault certificate set-attributes	Updates the specified attributes associated with the given certificate.
az keyvault certificate show	Gets information about a certificate.
az keyvault certificate show-deleted	Retrieves information about the specified deleted certificate.
az keyvault check-name	Checks that the managed hsm name is valid and is not already in

	use.
az keyvault create	Create a Vault or HSM.
az keyvault delete	Delete a Vault or HSM.
az keyvault delete-policy	Delete security policy settings for a Key Vault.
az keyvault key	Manage keys.
az keyvault key backup	Request that a backup of the specified key be downloaded to the client.
az keyvault key create	Create a new key, stores it, then returns key parameters and attributes to the client.
az keyvault key decrypt	Decrypt a single block of encrypted data.
az keyvault key delete	Delete a key of any type from storage in Vault or HSM.
az keyvault key download	Download the public part of a stored key.
az keyvault key encrypt	Encrypt an arbitrary sequence of bytes using an encryption key that is stored in a Vault or HSM.
az keyvault key get-policy-template	Return policy template as JSON encoded policy definition.
az keyvault key import	Import a private key.
az keyvault key list	List keys in the specified Vault or HSM.
az keyvault key list-deleted	List the deleted keys in the specified Vault or HSM.
az keyvault key list-versions	Retrieves a list of individual key versions with the same key name.
az keyvault key purge	Permanently delete the specified key.
az keyvault key random	Get the requested number of random bytes from a managed HSM.
az keyvault key recover	Recover the deleted key to its latest version.
az keyvault key restore	Restore a backed up key to a Vault or HSM.
az keyvault key rotate	Rotate the key based on the key policy by generating a new version of the key.
az keyvault key rotation-policy	Manage key's rotation policy.
az keyvault key rotation-policy show	Get the rotation policy of a Key Vault key.

az keyvault key rotation-policy update	Update the rotation policy of a Key Vault key.
az keyvault key set-attributes	The update key operation changes specified attributes of a stored key and can be applied to any key type and key version stored in Vault or HSM.
az keyvault key show	Get a key's attributes and, if it's an asymmetric key, its public material.
az keyvault key show-deleted	Get the public part of a deleted key.
az keyvault list	List Vaults and/or HSMs.
az keyvault list-deleted	Get information about the deleted Vaults or HSMs in a subscription.
az keyvault network-rule	Manage vault network ACLs.
az keyvault network-rule add	Add a network rule to the network ACLs for a Key Vault.
az keyvault network-rule list	List the network rules from the network ACLs for a Key Vault.
az keyvault network-rule remove	Remove a network rule from the network ACLs for a Key Vault.
az keyvault network-rule wait	Place the CLI in a waiting state until a condition of the vault is met.
az keyvault private-endpoint-connection	Manage vault/HSM private endpoint connections.
az keyvault private-endpoint-connection approve	Approve a private endpoint connection request for a Key Vault/HSM.
az keyvault private-endpoint-connection delete	Delete the specified private endpoint connection associated with a Key Vault/HSM.
az keyvault private-endpoint-connection list	List all private endpoint connections associated with a HSM.
az keyvault private-endpoint-connection reject	Reject a private endpoint connection request for a Key Vault/HSM.
az keyvault private-endpoint-connection show	Show details of a private endpoint connection associated with a Key Vault/HSM.
az keyvault private-endpoint-connection wait	Place the CLI in a waiting state until a condition of the private endpoint connection is met.
az keyvault private-link-resource	Manage vault/HSM private link resources.

az keyvault private-link-resource list	List the private link resources supported for a Key Vault/HSM.
az keyvault purge	Permanently delete the specified Vault or HSM. Aka Purges the deleted Vault or HSM.
az keyvault recover	Recover a Vault or HSM.
az keyvault region	Manage MHSM multi-regions.
az keyvault region add	Add regions for the managed HSM Pool.
az keyvault region list	Get regions information associated with the managed HSM Pool.
az keyvault region remove	Remove regions for the managed HSM Pool.
az keyvault region wait	Place the CLI in a waiting state until a condition of the HSM is met.
az keyvault restore	Manage full HSM restore.
az keyvault restore start	Restore a full backup of a HSM.
az keyvault role	Manage user roles for access control.
az keyvault role assignment	Manage role assignments.
az keyvault role assignment create	Create a new role assignment for a user, group, or service principal.
az keyvault role assignment delete	Delete a role assignment.
az keyvault role assignment list	List role assignments.
az keyvault role definition	Manage role definitions.
az keyvault role definition create	Create a custom role definition.
az keyvault role definition delete	Delete a role definition.
az keyvault role definition list	List role definitions.
az keyvault role definition show	Show the details of a role definition.
az keyvault role definition update	Update a role definition.
az keyvault secret	Manage secrets.

az keyvault secret backup	Backs up the specified secret.
az keyvault secret delete	Deletes a secret from a specified key vault.
az keyvault secret download	Download a secret from a KeyVault.
az keyvault secret list	List secrets in a specified key vault.
az keyvault secret list-deleted	Lists deleted secrets for the specified vault.
az keyvault secret list-versions	List all versions of the specified secret.
az keyvault secret purge	Permanently deletes the specified secret.
az keyvault secret recover	Recovers the deleted secret to the latest version.
az keyvault secret restore	Restores a backed up secret to a vault.
az keyvault secret set	Create a secret (if one doesn't exist) or update a secret in a KeyVault.
az keyvault secret set-attributes	Updates the attributes associated with a specified secret in a given key vault.
az keyvault secret show	Get a specified secret from a given key vault.
az keyvault secret show-deleted	Gets the specified deleted secret.
az keyvault security-domain	Manage security domain operations.
az keyvault security-domain download	Download the security domain file from the HSM.
az keyvault security-domain init-recovery	Retrieve the exchange key of the HSM.
az keyvault security-domain restore-blob	Enable to decrypt and encrypt security domain file as blob. Can be run in offline environment, before file is uploaded to HSM using security-domain upload.
az keyvault security-domain upload	Start to restore the HSM.
az keyvault security-domain wait	Place the CLI in a waiting state until HSM security domain operation is finished.
az keyvault set-policy	Update security policy settings for a Key Vault.
az keyvault show	Show details of a Vault or HSM.
az keyvault show-deleted	Show details of a deleted Vault or HSM.
az keyvault storage	Manage storage accounts.

az keyvault storage add	Creates or updates a new storage account.
az keyvault storage backup	Backs up the specified storage account.
az keyvault storage list	List storage accounts managed by the specified key vault.
az keyvault storage list-deleted	Lists deleted storage accounts for the specified vault.
az keyvault storage purge	Permanently deletes the specified storage account.
az keyvault storage recover	Recovers the deleted storage account.
az keyvault storage regenerate-key	Regenerates the specified key value for the given storage account.
az keyvault storage remove	Remove a Key Vault managed Azure Storage Account and all associated SAS definitions. This operation requires the storage/delete permission.
az keyvault storage restore	Restores a backed up storage account to a vault.
az keyvault storage sas-definition	Manage storage account SAS definitions.
az keyvault storage sas-definition create	Creates or updates a new SAS definition for the specified storage account.
az keyvault storage sas-definition delete	Deletes a SAS definition from a specified storage account.
az keyvault storage sas-definition list	List storage SAS definitions for the given storage account.
az keyvault storage sas-definition list-deleted	Lists deleted SAS definitions for the specified vault and storage account.
az keyvault storage sas-definition recover	Recovers the deleted SAS definition.
az keyvault storage sas-definition show	Gets information about a SAS definition for the specified storage account.
az keyvault storage sas-definition show-deleted	Gets the specified deleted sas definition.
az keyvault storage sas-definition update	Updates the specified attributes associated with the given SAS definition.
az keyvault storage show	Gets information about a specified storage account.
az keyvault storage show-	Gets the specified deleted storage account.

deleted	
az keyvault storage update	Updates the specified attributes associated with the given storage account.
az keyvault update	Update the properties of a Vault.
az keyvault update-hsm	Update the properties of a HSM.
az keyvault wait	Place the CLI in a waiting state until a condition of the Vault is met.
az keyvault wait-hsm	Place the CLI in a waiting state until a condition of the HSM is met.

az keyvault check-name

Edit

Checks that the managed hsm name is valid and is not already in use.

Azure CLI

```
az keyvault check-name --name
                       [--resource-type {hsm}]
```

Required Parameters

`--name -n`

The name of the HSM within the specified resource group.

Optional Parameters

`--resource-type`

Type of resource.

accepted values: hsm

default value: hsm

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault create

Edit

Create a Vault or HSM.

If `--enable-rbac-authorization` is not specified, then default permissions are created for the current user or service principal unless the `--no-self-perms` flag is specified.

Azure CLI

```
az keyvault create --resource-group
                   [--administrators]
                   [--bypass {AzureServices, None}]
                   [--default-action {Allow, Deny}]
                   [--enable-purge-protection {false, true}]
                   [--enable-rbac-authorization {false, true}]
```

```
                    [--enabled-for-deployment {false, true}]
                    [--enabled-for-disk-encryption {false, true}]
                    [--enabled-for-template-deployment {false, true}]
                    [--hsm-name]
                    [--location]
                    [--name]
                    [--network-acls]
                    [--network-acls-ips]
                    [--network-acls-vnets]
                    [--no-self-perms {false, true}]
                    [--no-wait]
                    [--public-network-access {Disabled, Enabled}]
                    [--retention-days]
                    [--sku]
                    [--tags]
```

Examples

Create a key vault with network ACLs specified (use --network-acls to specify IP and VNet rules by using a JSON string).

Azure CLI

```
az keyvault create --location westus2 --name MyKeyVault --resource-group
MyResourceGroup --network-acls "{\"ip\": [\"1.2.3.4\", \"2.3.4.0/24\"],
\"vnet\": [\"vnet_name_1/subnet_name1\", \"vnet_name_2/subnet_name2\",
\"/subscriptions/000000-0000-
0000/resourceGroups/MyResourceGroup/providers/Microsoft.Network/virtualNetwo
rks/MyVNet/subnets/MySubnet\"]}"
```

Create a key vault with network ACLs specified (use --network-acls to specify IP and VNet rules by using a JSON file).

Azure CLI

```
az keyvault create --location westus2 --name MyKeyVault --resource-group
MyResourceGroup --network-acls network-acls-example.json
```

Create a key vault with network ACLs specified (use --network-acls-ips to specify IP rules).

Azure CLI

```
az keyvault create --location westus2 --name MyKeyVault --resource-group
MyResourceGroup --network-acls-ips 3.4.5.0/24 4.5.6.0/24
```

Create a key vault with network ACLs specified (use --network-acls-vnets to specify VNet rules).

Azure CLI

```
az keyvault create --location westus2 --name MyKeyVault --resource-group
MyResourceGroup --network-acls-vnets vnet_name_2/subnet_name_2
vnet_name_3/subnet_name_3 /subscriptions/000000-0000-
0000/resourceGroups/MyResourceGroup/providers/Microsoft.Network/virtualNetwo
rks/vnet_name_4/subnets/subnet_name_4
```

Create a key vault with network ACLs specified (use --network-acls, --network-acls-ips and --network-acls-vnets together, redundant rules will be removed, finally there will be 4 IP rules and 3 VNet rules).

Azure CLI

```
az keyvault create --location westus2 --name MyKeyVault --resource-group
MyResourceGroup --network-acls "{\"ip\": [\"1.2.3.4\", \"2.3.4.0/24\"],
\"vnet\": [\"vnet_name_1/subnet_name1\", \"vnet_name_2/subnet_name2\"]}" --
network-acls-ips 3.4.5.0/24 4.5.6.0/24 --network-acls-vnets
vnet_name_2/subnet_name_2 vnet_name_3/subnet_name_3 /subscriptions/000000-
0000-
0000/resourceGroups/MyResourceGroup/providers/Microsoft.Network/virtualNetwo
rks/vnet_name_4/subnets/subnet_name_4
```

Create a key vault. (autogenerated)

Azure CLI

```
az keyvault create --location westus2 --name MyKeyVault --resource-group
MyResourceGroup
```

Required Parameters

`--resource-group -g`

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

Optional Parameters

`--administrators`

[HSM Only] Administrator role for data plane operations for Managed HSM. It accepts a space separated list of OIDs that will be assigned.

`--bypass`

Bypass traffic for space-separated uses.
accepted values: AzureServices, None

`--default-action`

Default action to apply when no rule matches.
accepted values: Allow, Deny

`--enable-purge-protection`

Property specifying whether protection against purge is enabled for this vault/managed HSM pool. Setting this property to true activates protection against purge for this vault/managed HSM pool and its content - only the Key Vault/Managed HSM service may initiate a hard, irrecoverable deletion. The setting is effective only if soft delete is also enabled. Enabling this functionality is irreversible.
accepted values: false, true

`--enable-rbac-authorization`

Property that controls how data actions are authorized. When true, the key vault will use Role Based Access Control (RBAC) for authorization of data actions, and the access policies specified in vault properties will be ignored. When false, the key vault will use the access policies specified in vault properties, and any policy stored on Azure Resource Manager will be ignored. If null or not specified, the vault is created with the default value of false. Note that management actions are always authorized with RBAC.
accepted values: false, true

`--enabled-for-deployment`

[Vault Only] Property to specify whether Azure Virtual Machines are permitted to retrieve certificates stored as secrets from the key vault.
accepted values: false, true

`--enabled-for-disk-encryption`

[Vault Only] Property to specify whether Azure Disk Encryption is permitted to retrieve secrets from the vault and unwrap keys.
accepted values: false, true

`--enabled-for-template-deployment`

[Vault Only] Property to specify whether Azure Resource Manager is permitted to retrieve secrets from the key vault.
accepted values: false, true

`--hsm-name`

Name of the HSM. (--hsm-name and --name/-n are mutually exclusive, please specify just one of them).

`--location -l`

Location. Values from: `az account list-locations`. You can configure the default location using `az configure --defaults location=<location>`.

`--name -n`

Name of the Vault.

`--network-acls`

Network ACLs. It accepts a JSON filename or a JSON string. JSON format: `{\"ip\": [<ip1>, <ip2>...],\"vnet\":[<vnet_name_1>/<subnet_name_1>,<subnet_id2>...]}`.

`--network-acls-ips`

Network ACLs IP rules. Space-separated list of IP addresses.

`--network-acls-vnets`

Network ACLS VNet rules. Space-separated list of Vnet/subnet pairs or subnet resource ids.

`--no-self-perms`

[Vault Only] Don't add permissions for the current user/service principal in the new vault.
accepted values: false, true

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

`--public-network-access`

Control permission for data plane traffic coming from public networks while private endpoint is enabled.
accepted values: Disabled, Enabled

`--retention-days`

Soft delete data retention days. It accepts >=7 and <=90.
default value: 90

`--sku`

Required. SKU details. Allowed values for Vault: premium, standard. Default: standard. Allowed values for HSM: Standard_B1, Custom_B32. Default: Standard_B1.

`--tags`

Space-separated tags: key[=value] [key[=value] ...]. Use "" to clear existing tags.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault delete

Edit

Delete a Vault or HSM.

Azure CLI

```
az keyvault delete [--hsm-name]
                   [--name]
                   [--no-wait]
                   [--resource-group]
```

Examples

Delete a key vault. (autogenerated)

Azure CLI

```
az keyvault delete --name MyKeyVault --resource-group MyResourceGroup
```

Optional Parameters

`--hsm-name`

Name of the HSM. (--hsm-name and --name/-n are mutually exclusive, please specify just one of them).

`--name -n`

Name of the Vault.

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault delete-policy

Edit

Delete security policy settings for a Key Vault.

Azure CLI

```
az keyvault delete-policy --name
                          [--application-id]
                          [--no-wait]
                          [--object-id]
                          [--resource-group]
                          [--spn]
                          [--upn]
```

Required Parameters

`--name -n`

Name of the Vault.

Optional Parameters

`--application-id`

Application ID of the client making request on behalf of a principal. Exposed for compound identity using on-behalf-of authentication flow.

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

`--object-id`

A GUID that identifies the principal that will receive permissions.

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

`--spn`

Name of a service principal that will receive permissions.

`--upn`

Name of a user principal that will receive permissions.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault list

Edit

List Vaults and/or HSMs.

Azure CLI

```
az keyvault list [--resource-group]
                 [--resource-type]
```

Optional Parameters

`--resource-group -g`

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

`--resource-type`

When --resource-type is not present the command will list all Vaults and HSMs. Possible values for --resource-type are vault and hsm.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault list-deleted

Edit

Get information about the deleted Vaults or HSMs in a subscription.

Azure CLI

```
az keyvault list-deleted [--resource-type]
```

Optional Parameters

`--resource-type`

When --resource-type is not present the command will list all deleted Vaults and HSMs. Possible values for --resource-type are vault and hsm.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault purge

Edit

Permanently delete the specified Vault or HSM. Aka Purges the deleted Vault or HSM.

Azure CLI

```
az keyvault purge [--hsm-name]
                  [--location]
                  [--name]
                  [--no-wait]
```

Optional Parameters

`--hsm-name`

Name of the deleted HSM. (--hsm-name and --name/-n are mutually exclusive, please specify just one of them).

`--location -l`

Location of the deleted Vault or HSM.

`--name -n`

Name of the deleted Vault.

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault recover

Edit

Recover a Vault or HSM.

Recover a previously deleted Vault or HSM for which soft delete was enabled.

Azure CLI

```
az keyvault recover [--hsm-name]
                    [--location]
                    [--name]
                    [--no-wait]
                    [--resource-group]
```

Examples

Recover a key vault. (autogenerated)

Azure CLI

```
az keyvault recover --location westus2 --name MyKeyVault --resource-group
MyResourceGroup
```

Optional Parameters

`--hsm-name`

Name of the deleted HSM. (--hsm-name and --name/-n are mutually exclusive, please specify just one of them).

`--location -l`

Location of the deleted Vault or HSM.

`--name -n`

Name of the deleted Vault.

`--no-wait`

Do not wait for the long-running operation to finish.

default value: False

`--resource-group -g`

Resource group of the deleted Vault or HSM.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault set-policy

Edit

Update security policy settings for a Key Vault.

Azure CLI

```
az keyvault set-policy --name
                       [--application-id]
                       [--certificate-permissions {all, backup, create,
delete, deleteissuers, get, getissuers, import, list, listissuers,
managecontacts, manageissuers, purge, recover, restore, setissuers, update}]
                       [--key-permissions {all, backup, create, decrypt,
delete, encrypt, get, getrotationpolicy, import, list, purge, recover,
release, restore, rotate, setrotationpolicy, sign, unwrapKey, update,
verify, wrapKey}]
                       [--no-wait]
                       [--object-id]
                       [--resource-group]
                       [--secret-permissions {all, backup, delete, get,
list, purge, recover, restore, set}]
                       [--spn]
                       [--storage-permissions {all, backup, delete,
deletesas, get, getsas, list, listsas, purge, recover, regeneratekey,
restore, set, setsas, update}]
                       [--upn]
```

Examples

Assign key permissions `get`, `list`, `import` and secret permissions `backup`, `restore` to an object id.

Azure CLI

```
az keyvault set-policy -n MyVault --key-permissions get list import --
secret-permissions backup restore --object-id {GUID}
```

Assign key permissions `get`, `list` to a UPN (User Principal Name).

Azure CLI

```
az keyvault set-policy -n MyVault --key-permissions get list --upn {UPN}
```

Assign key permissions `get`, `list` to a SPN (Service Principal Name).

Azure CLI

```
az keyvault set-policy -n MyVault --key-permissions get list --spn {SPN}
```

Required Parameters

`--name -n`

Name of the Vault.

Optional Parameters

`--application-id`

Application ID of the client making request on behalf of a principal. Exposed for compound identity using on-behalf-of authentication flow.

`--certificate-permissions`

Space-separated list of certificate permissions to assign.
accepted values: all, backup, create, delete, deleteissuers, get, getissuers, import, list, listissuers, managecontacts, manageissuers, purge, recover, restore, setissuers, update

`--key-permissions`

Space-separated list of key permissions to assign.
accepted values: all, backup, create, decrypt, delete, encrypt, get, getrotationpolicy, import, list, purge, recover, release, restore, rotate, setrotationpolicy, sign, unwrapKey, update, verify, wrapKey

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

`--object-id`

A GUID that identifies the principal that will receive permissions.

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

`--secret-permissions`

Space-separated list of secret permissions to assign.
accepted values: all, backup, delete, get, list, purge, recover, restore, set

`--spn`

Name of a service principal that will receive permissions.

`--storage-permissions`

Space-separated list of storage permissions to assign.
accepted values: all, backup, delete, deletesas, get, getsas, list, listsas, purge, recover, regeneratekey, restore, set, setsas, update

`--upn`

Name of a user principal that will receive permissions.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault show

Edit

Show details of a Vault or HSM.

Azure CLI

```
az keyvault show [--hsm-name]
                 [--name]
                 [--resource-group]
```

Examples

Show details of a key vault. (autogenerated)

Azure CLI

```
az keyvault show --name MyKeyVault
```

Optional Parameters

`--hsm-name`

Name of the HSM. (--hsm-name and --name/-n are mutually exclusive, please specify just one of them).

`--name -n`

Name of the Vault.

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault show-deleted

Edit

Show details of a deleted Vault or HSM.

Azure CLI

```
az keyvault show-deleted [--hsm-name]
                         [--location]
                         [--name]
```

Examples

Show details of a deleted key vault.

Azure CLI

```
az keyvault show-deleted --name MyKeyVault
```

Optional Parameters

`--hsm-name`

Name of the deleted HSM. (--hsm-name and --name/-n are mutually exclusive, please specify just one of them).

`--location -l`

Location of the deleted Vault or HSM.

`--name -n`

Name of the deleted Vault.

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault update

Edit

Update the properties of a Vault.

Azure CLI

```
az keyvault update --name
                   [--add]
                   [--bypass {AzureServices, None}]
                   [--default-action {Allow, Deny}]
                   [--enable-purge-protection {false, true}]
                   [--enable-rbac-authorization {false, true}]
                   [--enabled-for-deployment {false, true}]
                   [--enabled-for-disk-encryption {false, true}]
                   [--enabled-for-template-deployment {false, true}]
                   [--force-string]
                   [--no-wait]
                   [--public-network-access {Disabled, Enabled}]
                   [--remove]
                   [--resource-group]
                   [--retention-days]
                   [--set]
```

Examples

Update the properties of a Vault. (autogenerated)

Azure CLI

```
az keyvault update --enabled-for-disk-encryption true --name MyKeyVault --
resource-group MyResourceGroup
```

Required Parameters

`--name -n`

Name of the Vault.

Optional Parameters

`--add`

Add an object to a list of objects by specifying a path and key value pairs. Example: --add property.listProperty <key=value, string or JSON string>.
default value: []

`--bypass`

Bypass traffic for space-separated uses.
accepted values: AzureServices, None

`--default-action`

Default action to apply when no rule matches.
accepted values: Allow, Deny

`--enable-purge-protection`

Property specifying whether protection against purge is enabled for this vault/managed HSM pool. Setting this property to true activates protection against purge for this vault/managed HSM pool and its content - only the Key Vault/Managed HSM service may initiate a hard, irrecoverable deletion. The setting is effective only if soft delete is also enabled. Enabling this functionality is irreversible.
accepted values: false, true

`--enable-rbac-authorization`

Property that controls how data actions are authorized. When true, the key vault will use Role Based Access Control (RBAC) for authorization of data actions, and the access policies specified in vault properties will be ignored. When false, the key vault will use the access policies specified in vault properties, and any policy stored on Azure Resource Manager will be ignored. If null or not specified, the vault is created with the default value of false. Note that management actions are always authorized with RBAC.
accepted values: false, true

`--enabled-for-deployment`

[Vault Only] Property to specify whether Azure Virtual Machines are permitted to retrieve certificates stored as secrets from the key vault.
accepted values: false, true

`--enabled-for-disk-encryption`

[Vault Only] Property to specify whether Azure Disk Encryption is permitted to retrieve secrets from the vault and unwrap keys.
accepted values: false, true

`--enabled-for-template-deployment`

[Vault Only] Property to specify whether Azure Resource Manager is permitted to retrieve secrets from the key vault.
accepted values: false, true

`--force-string`

When using 'set' or 'add', preserve string literals instead of attempting to convert to JSON.
default value: False

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

`--public-network-access`

Control permission for data plane traffic coming from public networks while private endpoint is enabled.
accepted values: Disabled, Enabled

`--remove`

Remove a property or an element from a list. Example: --remove property.list OR --remove propertyToRemove.
default value: []

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

`--retention-days`

Soft delete data retention days. It accepts >=7 and <=90.

`--set`

Update an object by specifying a property path and value to set. Example: --set property1.property2=.

default value: []

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault update-hsm

Edit

Update the properties of a HSM.

Azure CLI

```
az keyvault update-hsm --hsm-name
                       [--add]
                       [--bypass {AzureServices, None}]
                       [--default-action {Allow, Deny}]
                       [--enable-purge-protection {false, true}]
                       [--force-string]
                       [--no-wait]
                       [--public-network-access {Disabled, Enabled}]
                       [--remove]
                       [--resource-group]
                       [--secondary-locations]
                       [--set]
```

Examples

Update the properties of a HSM.

Azure CLI

```
az keyvault update-hsm --enable-purge-protection true --hsm-name MyHSM --
resource-group MyResourceGroup
```

Required Parameters

`--hsm-name`

Name of the HSM.

Optional Parameters

`--add`

Add an object to a list of objects by specifying a path and key value pairs. Example: --add property.listProperty <key=value, string or JSON string>.

default value: []

`--bypass`

Bypass traffic for space-separated uses.
accepted values: AzureServices, None

`--default-action`

Default action to apply when no rule matches.
accepted values: Allow, Deny

`--enable-purge-protection -e`

Property specifying whether protection against purge is enabled for this vault/managed HSM pool. Setting this property to true activates protection against purge for this vault/managed HSM pool and its content - only the Key Vault/Managed HSM service may initiate a hard, irrecoverable deletion. The setting is effective only if soft delete is also enabled. Enabling this functionality is irreversible.
accepted values: false, true

`--force-string`

When using 'set' or 'add', preserve string literals instead of attempting to convert to JSON.
default value: False

`--no-wait`

Do not wait for the long-running operation to finish.
default value: False

`--public-network-access`

Control permission for data plane traffic coming from public networks while private endpoint is enabled.
accepted values: Disabled, Enabled

`--remove`

Remove a property or an element from a list. Example: --remove property.list OR --remove propertyToRemove.
default value: []

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

`--secondary-locations`

--secondary-locations extends/contracts an HSM pool to listed regions. The primary location where the resource was originally created CANNOT be removed.

`--set`

Update an object by specifying a property path and value to set. Example: --set property1.property2=.

default value: []

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault wait

Edit

Place the CLI in a waiting state until a condition of the Vault is met.

Azure CLI

```
az keyvault wait --name
                 [--created]
                 [--custom]
                 [--deleted]
                 [--exists]
                 [--interval]
                 [--resource-group]
                 [--timeout]
                 [--updated]
```

Examples

Pause CLI until the vault is created.

Azure CLI

```
az keyvault wait --name MyVault --created
```

Required Parameters

`--name -n`

Name of the Vault.

Optional Parameters

`--created`

Wait until created with 'provisioningState' at 'Succeeded'.

default value: False

`--custom`

Wait until the condition satisfies a custom JMESPath query. E.g. provisioningState!='InProgress', instanceView.statuses[?code=='PowerState/running'].

`--deleted`

Wait until deleted.
default value: False

`--exists`

Wait until the resource exists.
default value: False

`--interval`

Polling interval in seconds.
default value: 30

`--resource-group -g`

Proceed only if Key Vault belongs to the specified resource group.

`--timeout`

Maximum wait in seconds.
default value: 3600

`--updated`

Wait until updated with provisioningState at 'Succeeded'.
default value: False

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az keyvault wait-hsm

Edit

Place the CLI in a waiting state until a condition of the HSM is met.

Azure CLI

```
az keyvault wait-hsm --hsm-name
                     [--created]
                     [--custom]
                     [--deleted]
                     [--exists]
                     [--interval]
                     [--resource-group]
                     [--timeout]
                     [--updated]
```

Examples

Pause CLI until the HSM is created.

Azure CLI

```
az keyvault wait-hsm --hsm-name MyHSM --created
```

Required Parameters

`--hsm-name`

Name of the HSM.

Optional Parameters

`--created`

Wait until created with 'provisioningState' at 'Succeeded'.
default value: False

`--custom`

Wait until the condition satisfies a custom JMESPath query. E.g. provisioningState!='InProgress', instanceView.statuses[?code=='PowerState/running'].

`--deleted`

Wait until deleted.
default value: False

`--exists`

Wait until the resource exists.
default value: False

`--interval`

Polling interval in seconds.
default value: 30

`--resource-group -g`

Proceed only if HSM belongs to the specified resource group.

`--timeout`

Maximum wait in seconds.
default value: 3600

`--updated`

Wait until updated with provisioningState at 'Succeeded'.
default value: False

Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

Azure Key Vault key client library for .NET - version 4.5.0

Article • 03/21/2023

Azure Key Vault is a cloud service that provides secure storage of keys for encrypting your data. Multiple keys, and multiple versions of the same key, can be kept in the Azure Key Vault. Cryptographic keys in Azure Key Vault are represented as JSON Web Key (JWK) objects.

Azure Key Vault Managed HSM is a fully-managed, highly-available, single-tenant, standards-compliant cloud service that enables you to safeguard cryptographic keys for your cloud applications using FIPS 140-2 Level 3 validated HSMs.

The Azure Key Vault keys library client supports RSA keys and Elliptic Curve (EC) keys, each with corresponding support in hardware security modules (HSM). It offers operations to create, retrieve, update, delete, purge, backup, restore, and list the keys and its versions.

Source code | Package (NuGet) | API reference documentation | Product documentation | Samples | Migration guide

Getting started

Install the package

Install the Azure Key Vault keys client library for .NET with NuGet:

.NET CLI

```
dotnet add package Azure.Security.KeyVault.Keys
```

Prerequisites

- An Azure subscription.
- An existing Azure Key Vault. If you need to create an Azure Key Vault, you can use the Azure Portal or Azure CLI.
- Authorization to an existing Azure Key Vault using either RBAC (recommended) or access control.

If you are creating a standard Key Vault resource, run the following CLI command replacing `<your-resource-group-name>` and `<your-key-vault-name>` with your own, unique names:

PowerShell

```
az keyvault create --resource-group <your-resource-group-name> --name <your-key-vault-name>
```

If you are creating a Managed HSM resource, run the following CLI command:

PowerShell

```
az keyvault create --hsm-name <your-key-vault-name> --resource-group <your-resource-group-name> --administrators <your-user-object-id> --location <your-azure-location>
```

To get `<your-user-object-id>` you can run the following CLI command:

PowerShell

```
az ad user show --id <your-user-principal> --query id
```

Authenticate the client

In order to interact with the Azure Key Vault service, you'll need to create an instance of the KeyClient class. You need a **vault url**, which you may see as "DNS Name" in the portal, and credentials to instantiate a client object.

The examples shown below use a DefaultAzureCredential, which is appropriate for most scenarios including local development and production environments utilizing managed identity authentication. Additionally, we recommend using a managed identity for authentication in production environments. You can find more information on different ways of authenticating and their corresponding credential types in the Azure Identity documentation.

To use the `DefaultAzureCredential` provider shown below, or other credential providers provided with the Azure SDK, you must first install the Azure.Identity package:

.NET CLI

```
dotnet add package Azure.Identity
```

Activate your managed HSM

This section only applies if you are creating a Managed HSM. All data plane commands are disabled until the HSM is activated. You will not be able to create keys or assign roles. Only the designated administrators that were assigned during the create command can activate the HSM. To activate the HSM you must download the security domain.

To activate your HSM you need:

- Minimum 3 RSA key-pairs (maximum 10)
- Specify minimum number of keys required to decrypt the security domain (quorum)

To activate the HSM you send at least 3 (maximum 10) RSA public keys to the HSM. The HSM encrypts the security domain with these keys and sends it back. Once this security domain is successfully downloaded, your HSM is ready to use. You also need to specify quorum, which is the minimum number of private keys required to decrypt the security domain.

The example below shows how to use openssl to generate 3 self-signed certificate.

PowerShell

```
openssl req -newkey rsa:2048 -nodes -keyout cert_0.key -x509 -days 365 -out
cert_0.cer
openssl req -newkey rsa:2048 -nodes -keyout cert_1.key -x509 -days 365 -out
cert_1.cer
openssl req -newkey rsa:2048 -nodes -keyout cert_2.key -x509 -days 365 -out
cert_2.cer
```

Use the `az keyvault security-domain download` command to download the security domain and activate your managed HSM. The example below uses 3 RSA key pairs (only public keys are needed for this command) and sets the quorum to 2.

PowerShell

```
az keyvault security-domain download --hsm-name <your-key-vault-name> --sd-
wrapping-keys ./certs/cert_0.cer ./certs/cert_1.cer ./certs/cert_2.cer --sd-
quorum 2 --security-domain-file ContosoMHSM-SD.json
```

Create KeyClient

Instantiate a `DefaultAzureCredential` to pass to the client. The same instance of a token credential can be used with multiple clients if they will be authenticating with the same identity.

C#

```
// Create a new key client using the default credential from Azure.Identity
using environment variables previously set,
// including AZURE_CLIENT_ID, AZURE_CLIENT_SECRET, and AZURE_TENANT_ID.
var client = new KeyClient(vaultUri: new Uri(vaultUrl), credential: new
DefaultAzureCredential());

// Create a new key using the key client.
KeyVaultKey key = client.CreateKey("key-name", KeyType.Rsa);

// Retrieve a key using the key client.
key = client.GetKey("key-name");
```

Create CryptographyClient

Once you've created a `KeyVaultKey` in the Azure Key Vault, you can also create the CryptographyClient:

C#

```
// Create a new cryptography client using the same Key Vault or Managed HSM
endpoint, service version,
// and options as the KeyClient created earlier.
CryptographyClient cryptoClient = client.GetCryptographyClient(key.Name,
key.Properties.Version);
```

Key concepts

KeyVaultKey

Azure Key Vault supports multiple key types and algorithms, and enables the use of hardware security modules (HSM) for high value keys.

KeyClient

A `KeyClient` providing both synchronous and asynchronous operations exists in the SDK allowing for selection of a client based on an application's use case. Once you've

initialized a `KeyClient`, you can interact with the primary resource types in Azure Key Vault.

CryptographyClient

A `CryptographyClient` providing both synchronous and asynchronous operations exists in the SDK allowing for selection of a client based on an application's use case. Once you've initialized a `CryptographyClient`, you can use it to perform cryptographic operations with keys stored in Azure Key Vault.

Thread safety

We guarantee that all client instance methods are thread-safe and independent of each other (guideline). This ensures that the recommendation of reusing client instances is always safe, even across threads.

Additional concepts

Client options | Accessing the response | Long-running operations | Handling failures | Diagnostics | Mocking | Client lifetime

Examples

The Azure.Security.KeyVault.Keys package supports synchronous and asynchronous APIs.

The following section provides several code snippets using the `client` created above, covering some of the most common Azure Key Vault key service related tasks:

Sync examples

- Create a key
- Retrieve a key
- Update an existing key
- Delete a key
- Delete and purge a key
- List keys
- Encrypt and Decrypt

Async examples

- Create a key asynchronously
- List keys asynchronously
- Delete a key asynchronously

Create a key

Create a key to be stored in the Azure Key Vault. If a key with the same name already exists, then a new version of the key is created.

C#

```
// Create a key. Note that you can specify the type of key
// i.e. Elliptic curve, Hardware Elliptic Curve, RSA
KeyVaultKey key = client.CreateKey("key-name", KeyType.Rsa);

Console.WriteLine(key.Name);
Console.WriteLine(key.KeyType);

// Create a software RSA key
var rsaCreateKey = new CreateRsaKeyOptions("rsa-key-name",
hardwareProtected: false);
KeyVaultKey rsaKey = client.CreateRsaKey(rsaCreateKey);

Console.WriteLine(rsaKey.Name);
Console.WriteLine(rsaKey.KeyType);

// Create a hardware Elliptic Curve key
// Because only premium Azure Key Vault supports HSM backed keys , please
ensure your Azure Key Vault
// SKU is premium when you set "hardwareProtected" value to true
var echsmkey = new CreateEcKeyOptions("ec-key-name", hardwareProtected:
true);
KeyVaultKey ecKey = client.CreateEcKey(echsmkey);

Console.WriteLine(ecKey.Name);
Console.WriteLine(ecKey.KeyType);
```

Retrieve a key

`GetKey` retrieves a key previously stored in the Azure Key Vault.

C#

```
KeyVaultKey key = client.GetKey("key-name");

Console.WriteLine(key.Name);
Console.WriteLine(key.KeyType);
```

Update an existing key

`UpdateKeyProperties` updates a key previously stored in the Azure Key Vault.

C#

```
KeyVaultKey key = client.CreateKey("key-name", KeyType.Rsa);

// You can specify additional application-specific metadata in the form of
tags.
key.Properties.Tags["foo"] = "updated tag";

KeyVaultKey updatedKey = client.UpdateKeyProperties(key.Properties);

Console.WriteLine(updatedKey.Name);
Console.WriteLine(updatedKey.Properties.Version);
Console.WriteLine(updatedKey.Properties.UpdatedOn);
```

Delete a key

`StartDeleteKey` starts a long-running operation to delete a key previously stored in the Azure Key Vault. You can retrieve the key immediately without waiting for the operation to complete. When soft-delete is not enabled for the Azure Key Vault, this operation permanently deletes the key.

C#

```
DeleteKeyOperation operation = client.StartDeleteKey("key-name");

DeletedKey key = operation.Value;
Console.WriteLine(key.Name);
Console.WriteLine(key.DeletedOn);
```

Delete and purge a key

You will need to wait for the long-running operation to complete before trying to purge or recover the key.

C#

```
DeleteKeyOperation operation = client.StartDeleteKey("key-name");

// You only need to wait for completion if you want to purge or recover the
key.
while (!operation.HasCompleted)
{
```

```
    Thread.Sleep(2000);

    operation.UpdateStatus();
}

DeletedKey key = operation.Value;
client.PurgeDeletedKey(key.Name);
```

List Keys

This example lists all the keys in the specified Azure Key Vault.

C#

```
Pageable<KeyProperties> allKeys = client.GetPropertiesOfKeys();

foreach (KeyProperties keyProperties in allKeys)
{
    Console.WriteLine(keyProperties.Name);
}
```

Encrypt and Decrypt

This example creates a `CryptographyClient` and uses it to encrypt and decrypt with a key in Azure Key Vault.

C#

```
// Create a new cryptography client using the same Key Vault or Managed HSM
endpoint, service version,
// and options as the KeyClient created earlier.
var cryptoClient = client.GetCryptographyClient(key.Name,
key.Properties.Version);

byte[] plaintext = Encoding.UTF8.GetBytes("A single block of plaintext");

// encrypt the data using the algorithm RSAOAEP
EncryptResult encryptResult =
cryptoClient.Encrypt(EncryptionAlgorithm.RsaOaep, plaintext);

// decrypt the encrypted data.
DecryptResult decryptResult =
cryptoClient.Decrypt(EncryptionAlgorithm.RsaOaep, encryptResult.Ciphertext);
```

Create a key asynchronously

The asynchronous APIs are identical to their synchronous counterparts, but return with the typical "Async" suffix for asynchronous methods and return a Task.

C#

```
// Create a key of any type
KeyVaultKey key = await client.CreateKeyAsync("key-name", KeyType.Rsa);

Console.WriteLine(key.Name);
Console.WriteLine(key.KeyType);

// Create a software RSA key
var rsaCreateKey = new CreateRsaKeyOptions("rsa-key-name",
hardwareProtected: false);
KeyVaultKey rsaKey = await client.CreateRsaKeyAsync(rsaCreateKey);

Console.WriteLine(rsaKey.Name);
Console.WriteLine(rsaKey.KeyType);

// Create a hardware Elliptic Curve key
// Because only premium Azure Key Vault supports HSM backed keys , please
ensure your Azure Key Vault
// SKU is premium when you set "hardwareProtected" value to true
var echsmkey = new CreateEcKeyOptions("ec-key-name", hardwareProtected:
true);
KeyVaultKey ecKey = await client.CreateEcKeyAsync(echsmkey);

Console.WriteLine(ecKey.Name);
Console.WriteLine(ecKey.KeyType);
```

List keys asynchronously

Listing keys does not rely on awaiting the `GetPropertiesOfKeysAsync` method, but returns an `AsyncPageable<KeyProperties>` that you can use with the `await foreach` statement:

C#

```
AsyncPageable<KeyProperties> allKeys = client.GetPropertiesOfKeysAsync();

await foreach (KeyProperties keyProperties in allKeys)
{
    Console.WriteLine(keyProperties.Name);
}
```

Delete a key asynchronously

When deleting a key asynchronously before you purge it, you can await the `WaitForCompletionAsync` method on the operation. By default, this loops indefinitely but you can cancel it by passing a `CancellationToken`.

C#

```
DeleteKeyOperation operation = await client.StartDeleteKeyAsync("key-name");

// You only need to wait for completion if you want to purge or recover the
key.
await operation.WaitForCompletionAsync();

DeletedKey key = operation.Value;
await client.PurgeDeletedKeyAsync(key.Name);
```

Troubleshooting

See our troubleshooting guide for details on how to diagnose various failure scenarios.

General

When you interact with the Azure Key Vault key client library using the .NET SDK, errors returned by the service correspond to the same HTTP status codes returned for REST API requests.

For example, if you try to retrieve a key that doesn't exist in your Azure Key Vault, a `404` error is returned, indicating "Not Found".

C#

```
try
{
    KeyVaultKey key = client.GetKey("some_key");
}
catch (RequestFailedException ex)
{
    Console.WriteLine(ex.ToString());
}
```

You will notice that additional information is logged, like the client request ID of the operation.

text

```
Message:
    Azure.RequestFailedException : Service request failed.
    Status: 404 (Not Found)
Content:
    {"error":{"code":"KeyNotFound","message":"Key not found: some_key"}}

Headers:
    Cache-Control: no-cache
    Pragma: no-cache
    Server: Microsoft-IIS/10.0
    x-ms-keyvault-region: westus
    x-ms-request-id: 625f870e-10ea-41e5-8380-282e5cf768f2
    x-ms-keyvault-service-version: 1.1.0.866
    x-ms-keyvault-network-info:
addr=131.107.174.199;act_addr_fam=InterNetwork;
    X-AspNet-Version: 4.0.30319
    X-Powered-By: ASP.NET
    Strict-Transport-Security: max-age=31536000;includeSubDomains
    X-Content-Type-Options: nosniff
    Date: Tue, 18 Jun 2019 16:02:11 GMT
    Content-Length: 75
    Content-Type: application/json; charset=utf-8
    Expires: -1
```

Next steps

Several Azure Key Vault keys client library samples are available to you in this GitHub repository. These samples provide example code for additional scenarios commonly encountered while working with Azure Key Vault:

- Sample1_HelloWorld.md - for working with Azure Key Vault, including:
  - Create a key
  - Get an existing key
  - Update an existing key
  - Delete a key

- Sample2_BackupAndRestore.md - Contains the code snippets working with Azure Key Vault keys, including:
  - Backup and recover a key

- Sample3_GetKeys.md - Example code for working with Azure Key Vault keys, including:
  - Create keys
  - List all keys in the Key Vault
  - Update keys in the Key Vault
  - List versions of a specified key

    - Delete keys from the Key Vault
    - List deleted keys in the Key Vault

- Sample4_EncryptDecrypt.md - Example code for performing cryptographic operations with Azure Key Vault keys, including:
    - Encrypt and Decrypt data with the CryptographyClient

- Sample5_SignVerify.md - Example code for working with Azure Key Vault keys, including:
    - Sign a precalculated digest and verify the signature with Sign and Verify
    - Sign raw data and verify the signature with SignData and VerifyData

- Sample6_WrapUnwrap.md - Example code for working with Azure Key Vault keys, including:
    - Wrap and Unwrap a symmetric key

Additional Documentation

- For more extensive documentation on Azure Key Vault, see the API reference documentation.
- For Secrets client library see Secrets client library.
- For Certificates client library see Certificates client library.

Contributing

See the CONTRIBUTING.md for details on building, testing, and contributing to these libraries.

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit https://cla.microsoft.com.

When you submit a pull request, a CLA-bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., label, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repos using our CLA.

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

Azure Key Vault libraries for Java

Article • 03/21/2023

The Azure Key Vault client libraries for Java offer a convenient interface for making calls to Azure Key Vault. For more information about Azure Key Vault, see Introduction to Azure Key Vault.

Libraries for data access

The latest version of the Azure Key Vault libraries is version 4.x.x. Microsoft recommends using version 4.x.x for new applications.

If you cannot update existing applications to version 4.x.x, then Microsoft recommends using version 1.x.x.

Version 4.x.x

The version 4.x.x client libraries for Java are part of the Azure SDK for Java. The source code for the Azure Key Vault client libraries for Java is available on GitHub.

Use the following version 4.x.x libraries to work with certificates, keys, and secrets:

Library	Reference	Package	Source
azure-security-keyvault-certificates	Reference	Maven	GitHub
azure-security-keyvault-keys	Reference	Maven	GitHub
azure-security-keyvault-secrets	Reference	Maven	GitHub

Version 1.x.x

The source code for the Azure Key Vault client libraries for Java is available on GitHub.

Use the following version 1.x.x libraries to work with certificates, keys, and secrets:

Library	Reference	Package	Source
microsoft-azure-keyvault-core	Reference	Maven	GitHub
microsoft-azure-keyvault-cryptography	Reference	Maven	GitHub
microsoft-azure-keyvault-extensions	Reference	Maven	GitHub

Library	Reference	Package	Source
microsoft-azure-keyvault-webkey	Reference	Maven	GitHub
microsoft-azure-keyvault	Reference	Maven	GitHub

Libraries for resource management

Use the following library to work with the Azure Key Vault resource provider:

Library	Reference	Package	Source
azure-mgmt-keyvault	Reference	Maven	GitHub

Article • 04/26/2023

Reference	Package	Source
Key Vault - Administration	@azure/keyvault-admin	GitHub
Key Vault - Certificates	@azure/keyvault-certificates	GitHub
Common	@azure/keyvault-common	GitHub
Key Vault - Keys	@azure/keyvault-keys	GitHub
Key Vault - Secrets	@azure/keyvault-secrets	GitHub
Resource Management - Key Vault	@azure/arm-keyvault	GitHub
@azure/arm-keyvault-profile-2020-09-01-hybrid	@azure/arm-keyvault-profile-2020-09-01-hybrid	GitHub

Azure Key Vault Keys client library for Python - version 4.8.0

Article • 03/21/2023 • 7 minutes to read

Azure Key Vault helps solve the following problems:

- Cryptographic key management (this library) - create, store, and control access to the keys used to encrypt your data
- Secrets management (azure-keyvault-secrets) - securely store and control access to tokens, passwords, certificates, API keys, and other secrets
- Certificate management (azure-keyvault-certificates) - create, manage, and deploy public and private SSL/TLS certificates
- Vault administration (azure-keyvault-administration) - role-based access control (RBAC), and vault-level backup and restore options

Source code | Package (PyPI) | Package (Conda) | API reference documentation | Product documentation | Samples

Disclaimer

Azure SDK Python packages support for Python 2.7 has ended 01 January 2022. For more information and questions, please refer to https://github.com/Azure/azure-sdk-for-python/issues/20691.

Python 3.7 or later is required to use this package. For more details, please refer to Azure SDK for Python version support policy.

Getting started

Install the package

Install azure-keyvault-keys and azure-identity with pip:

Bash

```
pip install azure-keyvault-keys azure-identity
```

azure-identity is used for Azure Active Directory authentication as demonstrated below.

Prerequisites

- An Azure subscription
- Python 3.7 or later
- An existing Azure Key Vault. If you need to create one, you can do so using the Azure CLI by following the steps in this document.
- If using Managed HSM, an existing Key Vault Managed HSM. If you need to create a Managed HSM, you can do so using the Azure CLI by following the steps in this document.

Authenticate the client

In order to interact with the Azure Key Vault service, you will need an instance of a KeyClient, as well as a **vault url** and a credential object. This document demonstrates using a DefaultAzureCredential, which is appropriate for most scenarios, including local development and production environments. We recommend using a managed identity for authentication in production environments.

See azure-identity documentation for more information about other methods of authentication and their corresponding credential types.

Create a client

After configuring your environment for the DefaultAzureCredential to use a suitable method of authentication, you can do the following to create a key client (replacing the value of `VAULT_URL` with your vault's URL):

Python

```
VAULT_URL = os.environ["VAULT_URL"]
credential = DefaultAzureCredential()
client = KeyClient(vault_url=VAULT_URL, credential=credential)
```

> **NOTE:** For an asynchronous client, import `azure.keyvault.keys.aio`'s `KeyClient` instead.

Key concepts

Keys

Azure Key Vault can create and store RSA and elliptic curve keys. Both can optionally be protected by hardware security modules (HSMs). Azure Key Vault can also perform cryptographic operations with them. For more information about keys and supported operations and algorithms, see the Key Vault documentation.

KeyClient can create keys in the vault, get existing keys from the vault, update key metadata, and delete keys, as shown in the examples below.

Examples

This section contains code snippets covering common tasks:

- Create a key
- Retrieve a key
- Update an existing key
- Delete a key
- Configure automatic key rotation
- List keys
- Perform cryptographic operations
- Async API
- Asynchronously create a key
- Asynchronously list keys

Create a key

create_rsa_key and create_ec_key create RSA and elliptic curve keys in the vault, respectively. If a key with the same name already exists, a new version of that key is created.

Python

```python
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient

credential = DefaultAzureCredential()

key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)

# Create an RSA key
rsa_key = key_client.create_rsa_key("rsa-key-name", size=2048)
print(rsa_key.name)
print(rsa_key.key_type)

# Create an elliptic curve key
```

```
ec_key = key_client.create_ec_key("ec-key-name", curve="P-256")
print(ec_key.name)
print(ec_key.key_type)
```

Retrieve a key

get_key retrieves a key previously stored in the Vault.

Python

```python
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient

credential = DefaultAzureCredential()

key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)
key = key_client.get_key("key-name")
print(key.name)
```

Update an existing key

update_key_properties updates the properties of a key previously stored in the Key Vault.

Python

```python
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient

credential = DefaultAzureCredential()

key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)

# we will now disable the key for further use
updated_key = key_client.update_key_properties("key-name", enabled=False)

print(updated_key.name)
print(updated_key.properties.enabled)
```

Delete a key

begin_delete_key requests Key Vault delete a key, returning a poller which allows you to wait for the deletion to finish. Waiting is helpful when the vault has soft-delete

enabled, and you want to purge (permanently delete) the key as soon as possible. When soft-delete is disabled, `begin_delete_key` itself is permanent.

Python

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient

credential = DefaultAzureCredential()

key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)
deleted_key = key_client.begin_delete_key("key-name").result()

print(deleted_key.name)
print(deleted_key.deleted_date)
```

Configure automatic key rotation

update_key_rotation_policy allows you to configure automatic key rotation for a key by specifying a rotation policy. In addition, rotate_key allows you to rotate a key on-demand by creating a new version of the given key.

Python

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient, KeyRotationLifetimeAction,
KeyRotationPolicyAction

credential = DefaultAzureCredential()
key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)

# Set the key's automated rotation policy to rotate the key 30 days before
the key expires
actions = [KeyRotationLifetimeAction(KeyRotationPolicyAction.ROTATE,
time_before_expiry="P30D")]
# You may also specify the duration after which the newly rotated key will
expire
# In this example, any new key versions will expire after 90 days
updated_policy = key_client.update_key_rotation_policy("key-name",
expires_in="P90D", lifetime_actions=actions)

# You can get the current rotation policy for a key with
get_key_rotation_policy
current_policy = key_client.get_key_rotation_policy("key-name")

# Finally, you can rotate a key on-demand by creating a new version of the
key
rotated_key = key_client.rotate_key("key-name")
```

List keys

list_properties_of_keys lists the properties of all of the keys in the client's vault.

Python

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient

credential = DefaultAzureCredential()

key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)
keys = key_client.list_properties_of_keys()

for key in keys:
    # the list doesn't include values or versions of the keys
    print(key.name)
```

Cryptographic operations

CryptographyClient enables cryptographic operations (encrypt/decrypt, wrap/unwrap, sign/verify) using a particular key.

Python

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient
from azure.keyvault.keys.crypto import CryptographyClient,
EncryptionAlgorithm

credential = DefaultAzureCredential()
key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)

key = key_client.get_key("key-name")
crypto_client = CryptographyClient(key, credential=credential)
plaintext = b"plaintext"

result = crypto_client.encrypt(EncryptionAlgorithm.rsa_oaep, plaintext)
decrypted = crypto_client.decrypt(result.algorithm, result.ciphertext)
```

See the package documentation for more details of the cryptography API.

Async API

This library includes a complete set of async APIs. To use them, you must first install an async transport, such as aiohttp. See azure-core documentation for more information.

Async clients and credentials should be closed when they're no longer needed. These objects are async context managers and define async `close` methods. For example:

Python

```
from azure.identity.aio import DefaultAzureCredential
from azure.keyvault.keys.aio import KeyClient

credential = DefaultAzureCredential()

# call close when the client and credential are no longer needed
client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)
...
await client.close()
await credential.close()

# alternatively, use them as async context managers
(contextlib.AsyncExitStack can help)
client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)
async with client:
  async with credential:
    ...
```

Asynchronously create a key

create_rsa_key and create_ec_key create RSA and elliptic curve keys in the vault, respectively. If a key with the same name already exists, a new version of the key is created.

Python

```
from azure.identity.aio import DefaultAzureCredential
from azure.keyvault.keys.aio import KeyClient

credential = DefaultAzureCredential()
key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)

# Create an RSA key
rsa_key = await key_client.create_rsa_key("rsa-key-name", size=2048)
print(rsa_key.name)
print(rsa_key.key_type)
```

```
# Create an elliptic curve key
ec_key = await key_client.create_ec_key("ec-key-name", curve="P-256")
print(ec_key.name)
print(ec_key.key_type)
```

Asynchronously list keys

list_properties_of_keys lists the properties of all of the keys in the client's vault.

Python

```
from azure.identity.aio import DefaultAzureCredential
from azure.keyvault.keys.aio import KeyClient

credential = DefaultAzureCredential()
key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)
keys = key_client.list_properties_of_keys()

async for key in keys:
    print(key.name)
```

Troubleshooting

See the `azure-keyvault-keys` troubleshooting guide for details on how to diagnose various failure scenarios.

General

Key Vault clients raise exceptions defined in azure-core. For example, if you try to get a key that doesn't exist in the vault, KeyClient raises ResourceNotFoundError:

Python

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient
from azure.core.exceptions import ResourceNotFoundError

credential = DefaultAzureCredential()
key_client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential)

try:
    key_client.get_key("which-does-not-exist")
except ResourceNotFoundError as e:
    print(e.message)
```

Logging

This library uses the standard logging library for logging. Basic information about HTTP sessions (URLs, headers, etc.) is logged at INFO level.

Detailed DEBUG level logging, including request/response bodies and unredacted headers, can be enabled on a client with the `logging_enable` argument:

Python

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys import KeyClient
import sys
import logging

# Create a logger for the 'azure' SDK
logger = logging.getLogger('azure')
logger.setLevel(logging.DEBUG)

# Configure a console output
handler = logging.StreamHandler(stream=sys.stdout)
logger.addHandler(handler)

credential = DefaultAzureCredential()

# This client will log detailed information about its HTTP sessions, at
DEBUG level
client = KeyClient(vault_url="https://my-key-vault.vault.azure.net/",
credential=credential, logging_enable=True)
```

Similarly, `logging_enable` can enable detailed logging for a single operation, even when it isn't enabled for the client:

Python

```
client.get_key("my-key", logging_enable=True)
```

Next steps

Several samples are available in the Azure SDK for Python GitHub repository. These provide example code for additional Key Vault scenarios: | File | Description | |-----------|-------------| | hello_world.py (async version) | create/get/update/delete keys | | list_operations.py (async version) | basic list operations for keys | | backup_restore_operations.py (async version) | back up and recover keys | |

recover_purge_operations.py (async version) | recover and purge keys | |
send_request.py | use the `send_request` client method |

Additional documentation

For more extensive documentation on Azure Key Vault, see the API reference documentation.

Contributing

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit https://cla.microsoft.com.

When you submit a pull request, a CLA-bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., label, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repos using our CLA.

This project has adopted the Microsoft Open Source Code of Conduct. For more information, see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

Azure Key Vault REST API reference

Article • 04/18/2023

Use Key Vault to safeguard and manage cryptographic keys, certificates and secrets used by cloud applications and services.

Key Vault operations

Operation	Description
Check Name Availability	Checks that the vault name is valid and is not already in use.
Create Or Update	Create or update a key vault in the specified subscription.
Update Access Policy	Update access policies in a key vault in the specified subscription.
Get	Gets the specified Azure key vault.
List	The List operation gets information about the vaults associated with the subscription.
List By Resource Group	The List operation gets information about the vaults associated with the subscription and within the specified resource group.
List By Subscription	The List operation gets information about the vaults associated with the subscription.
Update	Update a key vault in the specified subscription.
Delete	Deletes the specified Azure key vault.
Get Deleted	Gets the deleted Azure key vault.
List Deleted	Gets information about the deleted vaults in a subscription.
Purge	Permanently deletes the specified vault.

Private link operations

Operation	Description
List By Vault	Gets the private link resources supported for the key vault.

Private endpoint connections operations

Operation	Description
Get	Gets the specified private endpoint connection associated with the key vault.
List By Resource	The List operation gets information about the private endpoint connections associated with the vault.
Put	Updates the specified private endpoint connection associated with the key vault.
Delete	Deletes the specified private endpoint connection associated with the key vault.

Managed HSM operations

Operation	Description
Create Or Update	Create or update a managed HSM Pool in the specified subscription.
Get	Gets the specified managed HSM Pool.
List By Resource Group	The List operation gets information about the managed HSM Pools associated with the subscription and within the specified resource group.
List By Subscription	The List operation gets information about the managed HSM Pools associated with the subscription.
Update	Update a managed HSM Pool in the specified subscription.
Get Deleted	Gets the specified deleted managed HSM.
List Deleted	The List operation gets information about the deleted managed HSMs associated with the subscription.
Delete	Deletes the specified managed HSM Pool.
Purge Deleted	Permanently deletes the specified managed HSM.

Private link operations

Operation	Description
List By MHSM Resource	Gets the private link resources supported for the managed HSM pool.

Private endpoint connections operations

Operation	Description
Get	Gets the specified private endpoint connection associated with the managed HSM Pool.
List By Resource	The List operation gets information about the private endpoint connections associated with the managed HSM Pool.
Put	Updates the specified private endpoint connection associated with the managed HSM Pool.
Delete	Deletes the specified private endpoint connection associated with the managed HSM Pool.

HSM Security Domain operations

Operation	Description
Download	Retrieves the Security Domain from the managed HSM. Calling this endpoint can be used to activate a provisioned managed HSM resource.
Download Pending	Retrieves the Security Domain download operation status.
Upload	Restore the provided Security Domain.
Upload Pending	Get Security Domain upload operation status.

Managed HSM Settings operations

Operation	Description
Get Setting	Get specified account setting object. Retrieves the setting object of a specified setting name.
Get Settings	List account settings. Retrieves a list of all the available account settings that can be configured.
Update Setting	Updates key vault account setting, stores it, then returns the setting name and value to the client. Description of the pool setting to be updated

Role-based access control operations

Role assignment operations

Operation	Description
Get	Get the specified role assignment.
List	Gets role assignments for a scope.
Create	Creates a role assignment.
Delete	Deletes a role assignment.

Role definition operations

Operation	Description
Get	Get the specified role definition.
List	Get all role definitions that are applicable at scope and above.
Create Or Update	Creates or updates a custom role definition.
Delete	Deletes a custom role definition.

Backup/restore operations

Operation	Description
Full Backup	Creates a full backup using a user-provided SAS token to an Azure blob storage container. This operation is supported only by the Managed HSM service.
Backup Status	Returns the status of full backup operation.
Full Restore	Restores all key materials using the SAS token pointing to a previously stored Azure Blob storage backup folder.
Selective Restore	Restores all key versions of a given key using user supplied SAS token pointing to a previously stored Azure Blob storage backup folder.
Restore Status	Returns the status of restore operation.

Key operations (Key Vault/Managed HSM)

Operation	Description

Operation	Description
Get Key	Gets the public part of a stored key.
Get Keys	List keys in the specified vault.
Get Key Versions	Retrieves a list of individual key versions with the same key name.
Create Key	Creates a new key, stores it, then returns key parameters and attributes to the client.
Import Key	Imports an externally created key, stores it, and returns key parameters and attributes to the client.
Update Key	The update key operation changes specified attributes of a stored key and can be applied to any key type and key version stored in Azure Key Vault.
Delete Key	Deletes a key of any type from storage in Azure Key Vault.
Get Deleted Key	Gets the public part of a deleted key.
Get Deleted Keys	Lists the deleted keys in the specified vault.
Purge Deleted Key	Permanently deletes the specified key.
Recover Deleted Key	Recovers the deleted key to its latest version.
Backup Key	Requests that a backup of the specified key be downloaded to the client.
Restore Key	Restores a backed up key to a vault.
Release Key	Releases a key. The release key operation is applicable to all key types. The target key must be marked exportable. This operation requires the keys/release permission.
Rotate Key	Creates a new key version, stores it, then returns key parameters, attributes and policy to the client. The operation will rotate the key based on the key policy. It requires the keys/rotate permission.
Get Key Rotation Policy	Lists the policy for a key. The GetKeyRotationPolicy operation returns the specified key policy resources in the specified key vault. This operation requires the keys/get permission.

Operation	Description
Update Key Rotation Policy	Updates the rotation policy for a key. Set specified members in the key policy. Leave others as undefined. This operation requires the keys/update permission.

Key operations (Managed HSM only)

Operation	Description
Get Random Bytes	Get the requested number of bytes containing random values from a managed HSM.

Cryptographic operations (Key Vault/Managed HSM)

Operation	Description
Decrypt	Decrypts a single block of encrypted data.
Encrypt	Encrypts an arbitrary sequence of bytes using an encryption key that is stored in a key vault.
Wrap Key	Wraps a symmetric key using a specified key.
Unwrap Key	Unwraps a symmetric key using the specified key that was initially used for wrapping that key.
Sign	Creates a signature from a digest using the specified key.
Verify	Verifies a signature using a specified key.

Secret operations (Key Vault only)

Operation	Description
Get Secret	Get a specified secret from a given key vault.
Get Secrets	List secrets in a specified key vault.
Get Secret Versions	List all versions of the specified secret.
Set Secret	Sets a secret in a specified key vault.
Update Secret	Updates the attributes associated with a specified secret in a given key vault.

Operation	Description
Delete Secret	Deletes a secret from a specified key vault.
Get Deleted Secret	Gets the specified deleted secret.
Get Deleted Secrets	Lists deleted secrets for the specified vault.
Purge Deleted Secret	Permanently deletes the specified secret.
Recover Deleted Secret	Recovers the deleted secret to the latest version.
Backup Secret	Backs up the specified secret.
Restore Secret	Restores a backed up secret to a vault.

Storage account key management operations (Key Vault only)

Storage Account configuration operations

Operation	Description
Get Storage Account	Gets information about a specified storage account. This operation requires the storage/get permission.
Get Storage Accounts	List storage accounts managed by the specified key vault. This operation requires the storage/list permission.
Update Storage Account	Updates the specified attributes associated with the given storage account. This operation requires the storage/set/update permission.
Set Storage Account	Creates or updates a new storage account. This operation requires the storage/set permission.
Delete Storage Account	Deletes a storage account. This operation requires the storage/delete permission.
Get Deleted Storage Account	Gets the specified deleted storage account.
Get Deleted Storage Accounts	Lists deleted storage accounts for the specified vault.
Purge Deleted Storage Account	Permanently deletes the specified storage account.

Operation	Description
Recover Deleted Storage Account	Recovers the deleted storage account.
Backup Storage Account	Backs up the specified storage account.
Restore Storage Account	Restores a backed-up storage account to a vault.

Storage Account key operations

Operation	Description
Regenerate Storage Account Key	Regenerates the specified key value for the given storage account. This operation requires the storage/regeneratekey permission.

Storage Account SAS operations

Operation	Description
Get Sas Definition	Gets information about a SAS definition for the specified storage account. This operation requires the storage/getsas permission.
Get Sas Definitions	List storage SAS definitions for the given storage account. This operation requires the storage/listsas permission.
Set Sas Definition	Creates or updates a new SAS definition for the specified storage account. This operation requires the storage/setsas permission.
Update Sas Definition	Updates the specified attributes associated with the given SAS definition. This operation requires the storage/setsas permission.
Delete Sas Definition	Deletes a SAS definition from a specified storage account. This operation requires the storage/deletesas permission.
Get Deleted Sas Definition	Gets the specified deleted sas definition.
Get Deleted Sas Definitions	Lists deleted SAS definitions for the specified vault and storage account.
Recover Deleted Sas Definition	Recovers the deleted SAS definition.

Certificate operations (Key Vault only)

Operation	Description
Get Certificate	Gets information about a certificate.
Get Certificates	List certificates in a specified key vault
Get Certificate Versions	List the versions of a certificate.
Create Certificate	Creates a new certificate.
Import Certificate	Imports a certificate into a specified key vault.
Merge Certificate	Merges a certificate or a certificate chain with a key pair existing on the server.
Get Certificate Operation	Gets the creation operation of a certificate.
Update Certificate Operation	Updates a certificate operation.
Delete Certificate Operation	Deletes the creation operation for a specific certificate.
Update Certificate	Updates the specified attributes associated with the given certificate.
Delete Certificate	Deletes a certificate from a specified key vault.
Get Deleted Certificate	Retrieves information about the specified deleted certificate.
Get Deleted Certificates	Lists the deleted certificates in the specified vault currently available for recovery.
Purge Deleted Certificate	Permanently deletes the specified deleted certificate.
Recover Deleted Certificate	Recovers the deleted certificate back to its current version under /certificates.
Backup Certificate	Backs up the specified certificate.
Restore Certificate	Restores a backed-up certificate to a vault.

Certificate policy operations

Operation	Description
Get Certificate Policy	Lists the policy for a certificate.

Operation	Description
Update Certificate Policy	Updates the policy for a certificate.

Certificate contacts operations

Operation	Description
Get Certificate Contacts	Lists the certificate contacts for a specified key vault.
Set Certificate Contacts	Sets the certificate contacts for the specified key vault.
Delete Certificate Contacts	Deletes the certificate contacts for a specified key vault.

Certificate issuer operations

Operation	Description
Get Certificate Issuer	Lists the specified certificate issuer.
Get Certificate Issuers	List certificate issuers for a specified key vault.
Set Certificate Issuer	Sets the specified certificate issuer.
Update Certificate Issuer	Updates the specified certificate issuer.
Delete Certificate Issuer	Deletes the specified certificate issuer.

See also

- For concepts and detailed information about Key Vault, see About Azure Key Vault.
- For concepts and detailed information about Managed HSM, see What is Azure Key Vault Managed HSM?
- For concepts and detailed information about data plane objects, see About keys, secrets, and certificates.
- For general information on constructing Azure REST API requests, see the Azure REST API reference
- For information specific to constructing Key Vault REST API requests, see
  - Common HTTP request parameters and headers
  - Authentication, requests and responses
- See the following topics for additional Key Vault concepts and details
  - Key Vault Developer's Guide
  - Key Vault versions

Microsoft.KeyVault resource types

Article • 12/28/2022 • 2 minutes to read

This article lists the available versions for each resource type.

For a list of changes in each API version, see change log

Resource types and versions

Types	Versions
managedHSMs	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview
managedHSMs/privateEndpointConnections	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview
vaults	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview 2019-09-01 2018-02-14 2018-02-14-preview 2016-10-01 2015-06-01
vaults/accessPolicies	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview 2019-09-01 2018-02-14 2018-02-14-preview 2016-10-01

Types	Versions
vaults/keys	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview 2019-09-01
vaults/keys/versions	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview 2019-09-01
vaults/privateEndpointConnections	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview 2019-09-01 2018-02-14
vaults/secrets	2022-07-01 2021-11-01-preview 2021-10-01 2021-06-01-preview 2021-04-01-preview 2020-04-01-preview 2019-09-01 2018-02-14 2018-02-14-preview 2016-10-01

Bring your own key specification

Article • 03/08/2023 • 5 minutes to read

This document describes specifications for importing HSM-protected keys from customers' on-premises HSMs into Key Vault.

Scenario

A Key Vault customer would like to securely transfer a key from their on-premises HSM outside Azure, into the HSM backing Azure Key Vault. The process of importing a key generated outside Key Vault is referred to as Bring Your Own Key (BYOK).

The following are the requirements:

- The key to be transferred never exists outside an HSM in plain text form.
- Outside an HSM, the key to be transferred is always protected by a key held in the Azure Key Vault HSM

Terminology

Key Name	Key Type	Origin	Description
Key Exchange Key (KEK)	RSA	Azure Key Vault HSM	An HSM backed RSA key pair generated in Azure Key Vault
Wrapping Key	AES	Vendor HSM	An [ephemeral] AES key generated by HSM on-premises
Target Key	RSA, EC, AES (Managed HSM only)	Vendor HSM	The key to be transferred to the Azure Key Vault HSM

Key Exchange Key: An HSM-backed key that customer generates in the key vault where the BYOK key will be imported. This KEK must have following properties:

- It's an RSA-HSM key (4096-bit or 3072-bit or 2048-bit)
- It will have fixed key_ops (ONLY 'import'), that will allow it to be used ONLY during BYOK
- Must be in the same vault where the Target Key will be imported

User steps

To perform a key transfer, a user performs following steps:

1. Generate KEK.
2. Retrieve the public key of the KEK.
3. Using HSM vendor provided BYOK tool - Import the KEK into the target HSM and exports the Target Key protected by the KEK.
4. Import the protected Target Key to Azure Key Vault.

Customers use the BYOK tool and documentation provided by HSM vendor to complete Steps 3. It produces a Key Transfer Blob (a ".byok" file).

HSM constraints

Existing HSM may apply constraints on key that they manage, including:

- The HSM may need to be configured to allow key wrap-based export
- The target key may need to be marked CKA_EXTRACTABLE for the HSM to allow controlled export
- In some cases, the KEK and wrapping key may need to be marked as CKA_TRUSTED, which allows it to be used to wrap keys in the HSM.

The configuration of source HSM is, generally, outside the scope of this specification. Microsoft expects the HSM vendor to produce documentation accompanying their BYOK tool to include any such configuration steps.

> ⓘ **Note**
>
> Several of these steps can be performed using other interfaces such as Azure PowerShell and Azure Portal. They can also be performed programmatically using equivalent functions in Key Vault SDK.

Generate KEK

Use the **az keyvault key create** command to create KEK with key operations set to import. Note down the key identifier 'kid' returned from the below command.

Azure CLI

```
az keyvault key create --kty RSA-HSM --size 4096 --name KEKforBYOK --ops
import --vault-name ContosoKeyVaultHSM
```

> ⓘ **Note**

Services support different KEK lengths; Azure SQL, for instance, only supports key lengths of **2048 or 3072 bytes**. Consult the documentation for your service for specifics.

Retrieve the public key of the KEK

Download the public key portion of the KEK and store it into a PEM file.

Azure CLI

```
az keyvault key download --name KEKforBYOK --vault-name ContosoKeyVaultHSM --file KEKforBYOK.publickey.pem
```

Generate key transfer blob using HSM vendor provided BYOK tool

Customer will use HSM Vendor provided BYOK tool to create a key transfer blob (stored as a ".byok" file). KEK public key (as a .pem file) will be one of the inputs to this tool.

Key Transfer Blob

Long term, Microsoft would like to use PKCS#11 CKM_RSA_AES_KEY_WRAP mechanism to transfer the target key to Azure Key Vault since this mechanism produces a single blob and, more importantly, the intermediate AES key is handled by the two HSMs and is guaranteed to be ephemeral. This mechanism is not presently available in some HSMs but the combination of protecting the target key with CKM_AES_KEY_WRAP_PAD using an AES key and protecting the AES key with CKM_RSA_PKCS_OAEP produces an equivalent blob.

The target key plaintext depends on the key type:

- For an RSA key, the private key ASN.1 DER encoding [RFC3447] wrapped in PKCS#8 [RFC5208]
- For an EC key, the private key ASN.1 DER encoding [RFC5915] wrapped in PKCS#8 [RFC5208]
- For an octet key, the raw bytes of the key

The bytes for the plaintext key are then transformed using the CKM_RSA_AES_KEY_WRAP mechanism:

- An ephemeral AES key is generated and encrypted with the wrapping RSA key using RSA-OAEP with SHA1.
- The encoded plaintext key is encrypted using the AES key using AES Key Wrap with Padding.
- The encrypted AES key and the encrypted plaintext key are concatenated to produce the final ciphertext blob.

The format of the transfer blob uses JSON Web Encryption compact serialization (RFC7516) primarily as a vehicle for delivering the required metadata to the service for correct decryption.

If CKM_RSA_AES_KEY_WRAP_PAD is used, the JSON serialization of the transfer blob would be:

JSON

```
{
  "schema_version": "1.0.0",
  "header":
  {
    "kid": "<key identifier of the KEK>",
    "alg": "dir",
    "enc": "CKM_RSA_AES_KEY_WRAP"
  },
  "ciphertext":"BASE64URL(<ciphertext contents>)",
  "generator": "BYOK tool name and version; source HSM name and firmware
version"
}
```

- kid = key identifier of KEK. For Key Vault keys it looks like this: https://ContosoKeyVaultHSM.vault.azure.net/keys/mykek/eba63d27e4e34e028839b53fac905621
- alg = algorithm.
- dir = Direct mode, that is, the referenced kid is used to directly protect the ciphertext that is an accurate representation of CKM_RSA_AES_KEY_WRAP
- generator = an informational field that denotes the name and version of BYOK tool and the source HSM manufacturer and model. This information is intended for use in troubleshooting and support.

The JSON blob is stored in a file with a ".byok" extension so that the Azure PowerShell/CLI clients treats it correctly when 'Add-AzKeyVaultKey' (PSH) or 'az keyvault key import' (CLI) commands are used.

Upload key transfer blob to import HSM-key

Customer will transfer the Key Transfer Blob (".byok" file) to an online workstation and then run a **az keyvault key import** command to import this blob as a new HSM-backed key into Key Vault.

To import an RSA key, use this command:

Azure CLI

```
az keyvault key import --vault-name ContosoKeyVaultHSM --name
ContosoFirstHSMkey --byok-file KeyTransferPackage-ContosoFirstHSMkey.byok --
ops encrypt decrypt
```

To import an EC key, you must specify key type and the curve name.

Azure CLI

```
az keyvault key import --vault-name ContosoKeyVaultHSM --name
ContosoFirstHSMkey --byok-file --kty EC-HSM --curve-name "P-256"
KeyTransferPackage-ContosoFirstHSMkey.byok --ops sign verify
```

When the above command is executed, it results in sending a REST API request as follows:

```
PUT https://contosokeyvaulthsm.vault.azure.net/keys/ContosoFirstHSMKey?api-
version=7.0
```

Request body when importing an RSA key:

JSON

```
{
  "key": {
    "kty": "RSA-HSM",
    "key_ops": [
      "decrypt",
      "encrypt"
    ],
    "key_hsm": "<Base64 encoded BYOK_BLOB>"
  },
  "attributes": {
    "enabled": true
  }
}
```

Request body when importing an EC key:

JSON

```
{
  "key": {
    "kty": "EC-HSM",
    "crv": "P-256",
    "key_ops": [
      "sign",
      "verify"
    ],
    "key_hsm": "<Base64 encoded BYOK_BLOB>"
  },
  "attributes": {
    "enabled": true
  }
}
```

“key_hsm” value is the entire contents of the KeyTransferPackage-ContosoFirstHSMkey.byok encoded in the Base64 format.

References

- Key Vault Developer's Guide

Next steps

- Step-by-step BYOK instructions: Import HSM-protected keys to Key Vault (BYOK)

Additional resources

Documentation

Key types, algorithms, and operations - Azure Key Vault

Supported key types, algorithms, and operations (details).

About keys - Azure Key Vault

Overview of Azure Key Vault REST interface and developer details for keys.

Overview of Key Management in Azure

This article provides an overview of Key Management in Azure.

How to generate & transfer HSM-protected keys – BYOK – Azure Key Vault

Use this article to help you plan for, generate, and transfer your own HSM-protected keys to use with Azure Key Vault. Also known as bring your own key (BYOK).

Configure key auto-rotation in Azure Key Vault Managed HSM

Use this guide to learn how to configure automated the rotation of a key in Azure Key Vault Managed HSM

Full backup/restore and selective restore for Azure Managed HSM

This document explains full backup/restore and selective restore

Enable multi-region replication on Azure Managed HSM (Preview)

Enable Multi-Region Replication on Azure Managed HSM (Preview)

Quickstart - Provision and activate an Azure Managed HSM

Quickstart showing how to provision and activate a managed HSM using Azure CLI

Show 5 more

Azure Policy built-in definitions for Key Vault

Article • 02/21/2023 • 15 minutes to read

This page is an index of Azure Policy built-in policy definitions for Key Vault. For additional Azure Policy built-ins for other services, see Azure Policy built-in definitions.

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the Azure Policy GitHub repo.

Key Vault (service)

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
[Preview]: [Preview]: Azure Key Vault Managed HSM should disable public network access	Disable public network access for your Azure Key Vault Managed HSM so that it's not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://docs.microsoft.com/azure/key-vault/managed-hsm/private-link#allow-trusted-services-to-access-managed-hsm.	Audit, Deny, Disabled	1.0.0-preview
[Preview]: [Preview]: Azure Key Vault Managed HSM should use private link	Private link provides a way to connect Azure Key Vault Managed HSM to your Azure resources without sending traffic over the public internet. Private link provides defense in depth protection against data exfiltration. Learn more at: https://docs.microsoft.com/azure/key-vault/managed-hsm/private-link	Audit, Disabled	1.0.0-preview
[Preview]: [Preview]: Azure Key Vaults should use private link	Azure Private Link lets you connect your virtual networks to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to key vault, you can reduce data leakage risks. Learn more about private links at: https://aka.ms/akvprivatelink.	Audit, Deny, Disabled	1.0.0-preview

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
[Preview]: [Preview]: Certificates should have the specified maximum validity period	Manage your organizational compliance requirements by specifying the maximum amount of time that a certificate can be valid within your key vault.	audit, Audit, deny, Deny, disabled, Disabled	2.2.0-preview
[Preview]: [Preview]: Certificates should not expire within the specified number of days	Manage certificates that will expire within a specified number of days to ensure your organization has sufficient time to rotate the certificate prior to expiration.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0-preview
[Preview]: [Preview]: Configure Azure Key Vault Managed HSM to disable public network access	Disable public network access for your Azure Key Vault Managed HSM so that it's not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://docs.microsoft.com/azure/key-vault/managed-hsm/private-link#allow-trusted-services-to-access-managed-hsm.	Modify, Disabled	2.0.0-preview
[Preview]: [Preview]: Configure Azure Key Vault Managed HSM with private endpoints	Private endpoints connect your virtual networks to Azure services without a public IP address at the source or destination. By mapping private endpoints to Azure Key Vault Managed HSM, you can reduce data leakage risks. Learn more at: https://docs.microsoft.com/azure/key-vault/managed-hsm/private-link.	DeployIfNotExists, Disabled	1.0.0-preview
[Preview]: [Preview]: Configure Azure Key Vaults with private endpoints	Private endpoints connect your virtual networks to Azure services without a public IP address at the source or destination. By mapping private endpoints to key vault, you can reduce data leakage risks. Learn more about private links at: https://aka.ms/akvprivatelink.	DeployIfNotExists, Disabled	1.0.0-preview

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
[Preview]: [Preview]: Private endpoint should be configured for Key Vault	Private link provides a way to connect Key Vault to your Azure resources without sending traffic over the public internet. Private link provides defense in depth protection against data exfiltration.	Audit, Deny, Disabled	1.1.0-preview
Azure Key Vault Managed HSM should have purge protection enabled	Malicious deletion of an Azure Key Vault Managed HSM can lead to permanent data loss. A malicious insider in your organization can potentially delete and purge Azure Key Vault Managed HSM. Purge protection protects you from insider attacks by enforcing a mandatory retention period for soft deleted Azure Key Vault Managed HSM. No one inside your organization or Microsoft will be able to purge your Azure Key Vault Managed HSM during the soft delete retention period.	Audit, Deny, Disabled	1.0.0
Azure Key Vault should disable public network access	Disable public network access for your key vault so that it's not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://aka.ms/akvprivatelink.	Audit, Deny, Disabled	1.0.0
Azure Key Vault should have firewall enabled	Enable the key vault firewall so that the key vault is not accessible by default to any public IPs. You can then configure specific IP ranges to limit access to those networks. Learn more at: https://docs.microsoft.com/azure/key-vault/general/network-security	Audit, Deny, Disabled	3.0.0
Certificates should be issued by the specified integrated certificate authority	Manage your organizational compliance requirements by specifying the Azure integrated certificate authorities that can issue certificates in your key vault such as Digicert or GlobalSign.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Certificates should be issued by the specified non-integrated certificate authority	Manage your organizational compliance requirements by specifying the custom or internal certificate authorities that can issue certificates in your key vault.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates should have the specified lifetime action triggers	Manage your organizational compliance requirements by specifying whether a certificate lifetime action is triggered at a specific percentage of its lifetime or at a certain number of days prior to its expiration.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates should use allowed key types	Manage your organizational compliance requirements by restricting the key types allowed for certificates.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates using elliptic curve cryptography should have allowed curve names	Manage the allowed elliptic curve names for ECC Certificates stored in key vault. More information can be found at https://aka.ms/akvpolicy .	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates using RSA cryptography should have the specified minimum key size	Manage your organizational compliance requirements by specifying a minimum key size for RSA certificates stored in your key vault.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Configure key vaults to enable firewall	Enable the key vault firewall so that the key vault is not accessible by default to any public IPs. You can then configure specific IP ranges to limit access to those networks. Learn more at: https://docs.microsoft.com/azure/key-vault/general/network-security	Modify, Disabled	1.1.1

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Deploy - Configure diagnostic settings for Azure Key Vault to Log Analytics workspace	Deploys the diagnostic settings for Azure Key Vault to stream resource logs to a Log Analytics workspace when any Key Vault which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	2.0.1
Deploy - Configure diagnostic settings to a Log Analytics workspace to be enabled on Azure Key Vault Managed HSM	Deploys the diagnostic settings for Azure Key Vault Managed HSM to stream to a regional Log Analytics workspace when any Azure Key Vault Managed HSM which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	1.0.0
Deploy - Configure diagnostic settings to an Event Hub to be enabled on Azure Key Vault Managed HSM	Deploys the diagnostic settings for Azure Key Vault Managed HSM to stream to a regional Event Hub when any Azure Key Vault Managed HSM which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	1.0.0
Deploy Diagnostic Settings for Key Vault to Event Hub	Deploys the diagnostic settings for Key Vault to stream to a regional Event Hub when any Key Vault which is missing this diagnostic settings is created or updated.	deployIfNotExists	3.0.0
Deploy Diagnostic Settings for Key Vault to Log Analytics workspace	Deploys the diagnostic settings for Key Vault to stream to a regional Log Analytics workspace when any Key Vault which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	3.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Key Vault keys should have an expiration date	Cryptographic keys should have a defined expiration date and not be permanent. Keys that are valid forever provide a potential attacker with more time to compromise the key. It is a recommended security practice to set expiration dates on cryptographic keys.	Audit, Deny, Disabled	1.0.2
Key Vault secrets should have an expiration date	Secrets should have a defined expiration date and not be permanent. Secrets that are valid forever provide a potential attacker with more time to compromise them. It is a recommended security practice to set expiration dates on secrets.	Audit, Deny, Disabled	1.0.2
Key Vault should use a virtual network service endpoint	This policy audits any Key Vault not configured to use a virtual network service endpoint.	Audit, Disabled	1.0.0
Key vaults should have purge protection enabled	Malicious deletion of a key vault can lead to permanent data loss. A malicious insider in your organization can potentially delete and purge key vaults. Purge protection protects you from insider attacks by enforcing a mandatory retention period for soft deleted key vaults. No one inside your organization or Microsoft will be able to purge your key vaults during the soft delete retention period.	Audit, Deny, Disabled	2.0.0
Key vaults should have soft delete enabled	Deleting a key vault without soft delete enabled permanently deletes all secrets, keys, and certificates stored in the key vault. Accidental deletion of a key vault can lead to permanent data loss. Soft delete allows you to recover an accidentally deleted key vault for a configurable retention period.	Audit, Deny, Disabled	3.0.0
Keys should be backed by a hardware security module (HSM)	An HSM is a hardware security module that stores keys. An HSM provides a physical layer of protection for cryptographic keys. The cryptographic key cannot leave a physical HSM which provides a greater level of security than a software key.	Audit, Deny, Disabled	1.0.1

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Keys should be the specified cryptographic type RSA or EC	Some applications require the use of keys backed by a specific cryptographic type. Enforce a particular cryptographic key type, RSA or EC, in your environment.	Audit, Deny, Disabled	1.0.1
Keys should have more than the specified number of days before expiration	If a key is too close to expiration, an organizational delay to rotate the key may result in an outage. Keys should be rotated at a specified number of days prior to expiration to provide sufficient time to react to a failure.	Audit, Deny, Disabled	1.0.1
Keys should have the specified maximum validity period	Manage your organizational compliance requirements by specifying the maximum amount of time in days that a key can be valid within your key vault.	Audit, Deny, Disabled	1.0.1
Keys should not be active for longer than the specified number of days	Specify the number of days that a key should be active. Keys that are used for an extended period of time increase the probability that an attacker could compromise the key. As a good security practice, make sure that your keys have not been active longer than two years.	Audit, Deny, Disabled	1.0.1
Keys using elliptic curve cryptography should have the specified curve names	Keys backed by elliptic curve cryptography can have different curve names. Some applications are only compatible with specific elliptic curve keys. Enforce the types of elliptic curve keys that are allowed to be created in your environment.	Audit, Deny, Disabled	1.0.1
Keys using RSA cryptography should have a specified minimum key size	Set the minimum allowed key size for use with your key vaults. Use of RSA keys with small key sizes is not a secure practice and doesn't meet many industry certification requirements.	Audit, Deny, Disabled	1.0.1

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Resource logs in Azure Key Vault Managed HSM should be enabled	To recreate activity trails for investigation purposes when a security incident occurs or when your network is compromised, you may want to audit by enabling resource logs on Managed HSMs. Please follow the instructions here: https://docs.microsoft.com/azure/key-vault/managed-hsm/logging.	AuditIfNotExists, Disabled	1.0.0
Resource logs in Key Vault should be enabled	Audit enabling of resource logs. This enables you to recreate activity trails to use for investigation purposes when a security incident occurs or when your network is compromised	AuditIfNotExists, Disabled	5.0.0
Secrets should have content type set	A content type tag helps identify whether a secret is a password, connection string, etc. Different secrets have different rotation requirements. Content type tag should be set on secrets.	Audit, Deny, Disabled	1.0.1
Secrets should have more than the specified number of days before expiration	If a secret is too close to expiration, an organizational delay to rotate the secret may result in an outage. Secrets should be rotated at a specified number of days prior to expiration to provide sufficient time to react to a failure.	Audit, Deny, Disabled	1.0.1
Secrets should have the specified maximum validity period	Manage your organizational compliance requirements by specifying the maximum amount of time in days that a secret can be valid within your key vault.	Audit, Deny, Disabled	1.0.1
Secrets should not be active for longer than the specified number of days	If your secrets were created with an activation date set in the future, you must ensure that your secrets have not been active for longer than the specified duration.	Audit, Deny, Disabled	1.0.1

Key Vault (objects)

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
[Preview]: [Preview]: Certificates should have the specified maximum validity period	Manage your organizational compliance requirements by specifying the maximum amount of time that a certificate can be valid within your key vault.	audit, Audit, deny, Deny, disabled, Disabled	2.2.0-preview
[Preview]: [Preview]: Certificates should not expire within the specified number of days	Manage certificates that will expire within a specified number of days to ensure your organization has sufficient time to rotate the certificate prior to expiration.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0-preview
Certificates should be issued by the specified integrated certificate authority	Manage your organizational compliance requirements by specifying the Azure integrated certificate authorities that can issue certificates in your key vault such as Digicert or GlobalSign.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates should be issued by the specified non-integrated certificate authority	Manage your organizational compliance requirements by specifying the custom or internal certificate authorities that can issue certificates in your key vault.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates should have the specified lifetime action triggers	Manage your organizational compliance requirements by specifying whether a certificate lifetime action is triggered at a specific percentage of its lifetime or at a certain number of days prior to its expiration.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates should use allowed key types	Manage your organizational compliance requirements by restricting the key types allowed for certificates.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Certificates using elliptic curve cryptography should have allowed curve names	Manage the allowed elliptic curve names for ECC Certificates stored in key vault. More information can be found at https://aka.ms/akvpolicy.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Certificates using RSA cryptography should have the specified minimum key size	Manage your organizational compliance requirements by specifying a minimum key size for RSA certificates stored in your key vault.	audit, Audit, deny, Deny, disabled, Disabled	2.1.0
Key Vault keys should have an expiration date	Cryptographic keys should have a defined expiration date and not be permanent. Keys that are valid forever provide a potential attacker with more time to compromise the key. It is a recommended security practice to set expiration dates on cryptographic keys.	Audit, Deny, Disabled	1.0.2
Key Vault secrets should have an expiration date	Secrets should have a defined expiration date and not be permanent. Secrets that are valid forever provide a potential attacker with more time to compromise them. It is a recommended security practice to set expiration dates on secrets.	Audit, Deny, Disabled	1.0.2
Keys should be backed by a hardware security module (HSM)	An HSM is a hardware security module that stores keys. An HSM provides a physical layer of protection for cryptographic keys. The cryptographic key cannot leave a physical HSM which provides a greater level of security than a software key.	Audit, Deny, Disabled	1.0.1
Keys should be the specified cryptographic type RSA or EC	Some applications require the use of keys backed by a specific cryptographic type. Enforce a particular cryptographic key type, RSA or EC, in your environment.	Audit, Deny, Disabled	1.0.1
Keys should have more than the specified number of days before expiration	If a key is too close to expiration, an organizational delay to rotate the key may result in an outage. Keys should be rotated at a specified number of days prior to expiration to provide sufficient time to react to a failure.	Audit, Deny, Disabled	1.0.1

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Keys should have the specified maximum validity period	Manage your organizational compliance requirements by specifying the maximum amount of time in days that a key can be valid within your key vault.	Audit, Deny, Disabled	1.0.1
Keys should not be active for longer than the specified number of days	Specify the number of days that a key should be active. Keys that are used for an extended period of time increase the probability that an attacker could compromise the key. As a good security practice, make sure that your keys have not been active longer than two years.	Audit, Deny, Disabled	1.0.1
Keys using elliptic curve cryptography should have the specified curve names	Keys backed by elliptic curve cryptography can have different curve names. Some applications are only compatible with specific elliptic curve keys. Enforce the types of elliptic curve keys that are allowed to be created in your environment.	Audit, Deny, Disabled	1.0.1
Keys using RSA cryptography should have a specified minimum key size	Set the minimum allowed key size for use with your key vaults. Use of RSA keys with small key sizes is not a secure practice and doesn't meet many industry certification requirements.	Audit, Deny, Disabled	1.0.1
Secrets should have content type set	A content type tag helps identify whether a secret is a password, connection string, etc. Different secrets have different rotation requirements. Content type tag should be set on secrets.	Audit, Deny, Disabled	1.0.1
Secrets should have more than the specified number of days before expiration	If a secret is too close to expiration, an organizational delay to rotate the secret may result in an outage. Secrets should be rotated at a specified number of days prior to expiration to provide sufficient time to react to a failure.	Audit, Deny, Disabled	1.0.1
Secrets should have the specified maximum validity period	Manage your organizational compliance requirements by specifying the maximum amount of time in days that a secret can be valid within your key vault.	Audit, Deny, Disabled	1.0.1

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Secrets should not be active for longer than the specified number of days	If your secrets were created with an activation date set in the future, you must ensure that your secrets have not been active for longer than the specified duration.	Audit, Deny, Disabled	1.0.1

Next steps

- See the built-ins on the Azure Policy GitHub repo.
- Review the Azure Policy definition structure.
- Review Understanding policy effects.

Additional resources

Documentation

Integrate Azure Key Vault with Azure Policy

Learn how to integrate Azure Key Vault with Azure Policy

Enable Azure Key Vault logging

How to enable logging for Azure Key Vault, which saves information in an Azure storage account that you provide.

Enable soft-delete on all key vault objects - Azure Key Vault

Use this document to adopt soft-delete for all key vaults and to make application and administration changes to avoid conflict errors.

Troubleshoot issues with implementing Azure policy on Key Vault

Troubleshooting issues with implementing Azure policy on Key Vault

Configure cryptographic key auto-rotation in Azure Key Vault

Use this guide to learn how to configure automated the rotation of a key in Azure Key Vault

Migrate to Azure role-based access control

Learn how to migrate from vault access policies to Azure roles.

Azure Key Vault logging

Learn how to monitor access to your key vaults by enabling logging for Azure Key Vault, which saves information in an Azure storage account that you provide.

Configure key auto-rotation in Azure Key Vault Managed HSM

Use this guide to learn how to configure automated the rotation of a key in Azure Key Vault

Managed HSM

Show 5 more

Training

Learning paths and modules

Deploy and secure Azure Key Vault - Training

Protect your keys, certificates, and secrets in Azure Key Vault. Learn to configure key vault for the most secure deployment.

Learning certificate

Microsoft Certified: Azure Administrator Associate - Certifications

Azure administrators implement, manage, and monitor an organization's Microsoft Azure environment, including virtual networks, storage, compute, identity, security, and governance.

Azure Key Vault service limits

Article • 04/20/2021 • 4 minutes to read

Azure Key Vault service supports two resource types: Vaults and Managed HSMs. The following two sections describe the service limits for each of them respectively.

Resource type: vault

This section describes service limits for resource type `vaults`.

Key transactions (maximum transactions allowed in 10 seconds, per vault per region[1]):

Key type	HSM key CREATE key	HSM key All other transactions	Software key CREATE key	Software key All other transactions
RSA 2,048-bit	10	2,000	20	4,000
RSA 3,072-bit	10	500	20	1,000
RSA 4,096-bit	10	250	20	500
ECC P-256	10	2,000	20	4,000
ECC P-384	10	2,000	20	4,000
ECC P-521	10	2,000	20	4,000
ECC SECP256K1	10	2,000	20	4,000

> **Note**
>
> In the previous table, we see that for RSA 2,048-bit software keys, 4,000 GET transactions per 10 seconds are allowed. For RSA 2,048-bit HSM-keys, 2,000 GET transactions per 10 seconds are allowed.
>
> The throttling thresholds are weighted, and enforcement is on their sum. For example, as shown in the previous table, when you perform GET operations on RSA HSM-keys, it's eight times more expensive to use 4,096-bit keys compared to 2,048-bit keys. That's because 2,000/250 = 8.

In a given 10-second interval, an Azure Key Vault client can do *only one* of the following operations before it encounters a `429` throttling HTTP status code:

- 4,000 RSA 2,048-bit software-key GET transactions
- 2,000 RSA 2,048-bit HSM-key GET transactions
- 250 RSA 4,096-bit HSM-key GET transactions
- 248 RSA 4,096-bit HSM-key GET transactions and 16 RSA 2,048-bit HSM-key GET transactions

Secrets, managed storage account keys, and vault transactions:

Transactions type	Maximum transactions allowed in 10 seconds, per vault per region[1]
All transactions	4,000

For information on how to handle throttling when these limits are exceeded, see Azure Key Vault throttling guidance.

[1] A subscription-wide limit for all transaction types is five times per key vault limit.

Backup keys, secrets, certificates

When you back up a key vault object, such as a secret, key, or certificate, the backup operation will download the object as an encrypted blob. This blob cannot be decrypted outside of Azure. To get usable data from this blob, you must restore the blob into a key vault within the same Azure subscription and Azure geography

Transactions type	Maximum key vault object versions allowed
Back up individual key, secret, certificate	500

> **Note**
>
> Attempting to backup a key, secret, or certificate object with more versions than above limit will result in an error. It is not possible to delete previous versions of a key, secret, or certificate.

Limits on count of keys, secrets and certificates:

Key Vault does not restrict the number of keys, secrets or certificates that can be stored in a vault. The transaction limits on the vault should be taken into account to ensure that operations are not throttled.

Key Vault does not restrict the number of versions on a secret, key or certificate, but storing a large number of versions (500+) can impact the performance of backup operations. See Azure Key Vault Backup.

Resource type: Managed HSM

This section describes service limits for resource type `managed HSM`.

Object limits

Item	Limits
Number of HSM instances per subscription per region	5
Number of keys per HSM instance	5000
Number of versions per key	100
Number of custom role definitions per HSM instance	50
Number of role assignments at HSM scope	50
Number of role assignments at each individual key scope	10

Transaction limits for administrative operations (number of operations per second per HSM instance)

Operation	Number of operations per second
All RBAC operations (includes all CRUD operations for role definitions and role assignments)	5
Full HSM Backup/Restore (only one concurrent backup or restore operation per HSM instance supported)	1

Transaction limits for cryptographic operations (number of operations per second per HSM instance)

- Each Managed HSM instance constitutes three load balanced HSM partitions. The throughput limits are a function of underlying hardware capacity allocated for each partition. The tables below show maximum throughput with at least one partition available. Actual throughput may be up to 3x higher if all three partitions are available.
- Throughput limits noted assume that one single key is being used to achieve maximum throughput. For example, if a single RSA-2048 key is used the maximum throughput will be 1100 sign operations. If you use 1100 different keys with one transaction per second each, they will not be able to achieve the same throughput.

RSA key operations (number of operations per second per HSM instance)

Operation	2048-bit	3072-bit	4096-bit
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	10000	10000	6000
Decrypt	1100	360	160
Wrap	10000	10000	6000
Unwrap	1100	360	160
Sign	1100	360	160
Verify	10000	10000	6000

EC key operations (number of operations per second per HSM instance)

This table describes number of operations per second for each curve type.

Operation	P-256	P-256K	P-384	P-521
Create Key	1	1	1	1
Delete Key (soft-delete)	10	10	10	10
Purge Key	10	10	10	10
Backup Key	10	10	10	10
Restore Key	10	10	10	10
Get Key Information	1100	1100	1100	1100
Sign	260	260	165	56
Verify	130	130	82	28

AES key operations (number of operations per second per HSM instance)

- Encrypt and Decrypt operations assume a 4KB packet size.
- Throughput limits for Encrypt/Decrypt apply to AES-CBC and AES-GCM algorithms.
- Throughput limits for Wrap/Unwrap apply to AES-KW algorithm.

Operation	128-bit	192-bit	256-bit
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	8000	8000	8000
Decrypt	8000	8000	8000
Wrap	9000	9000	9000
Unwrap	9000	9000	9000

Additional resources

Documentation

Azure Key Vault throttling guidance

Key Vault throttling limits the number of concurrent calls to prevent overuse of resources.

Configure cryptographic key auto-rotation in Azure Key Vault

Use this guide to learn how to configure automated the rotation of a key in Azure Key Vault

About keys - Azure Key Vault

Overview of Azure Key Vault REST interface and developer details for keys.

Key types, algorithms, and operations - Azure Key Vault

Supported key types, algorithms, and operations (details).

Import Key - Import Key - REST API (Azure Key Vault)

Imports an externally created key, stores it, and returns key parameters and attributes to the client.

Best practices for secrets management - Azure Key Vault

Learn about best practices for Azure Key Vault secrets management.

Azure Policy Regulatory Compliance controls for Azure Key Vault

Lists Azure Policy Regulatory Compliance controls available for Azure Key Vault. These built-in policy definitions provide common approaches to managing the compliance of your Azure resources.

About Azure Key Vault secrets - Azure Key Vault

Overview of Azure Key Vault secrets.

Show 5 more