### **Agenda:** **Key Vault**

### Secrets vs Keys

### Accessing Keys and Secrets

### Use cases for Key Vault

### Use Azure Key Vault from a Web Application

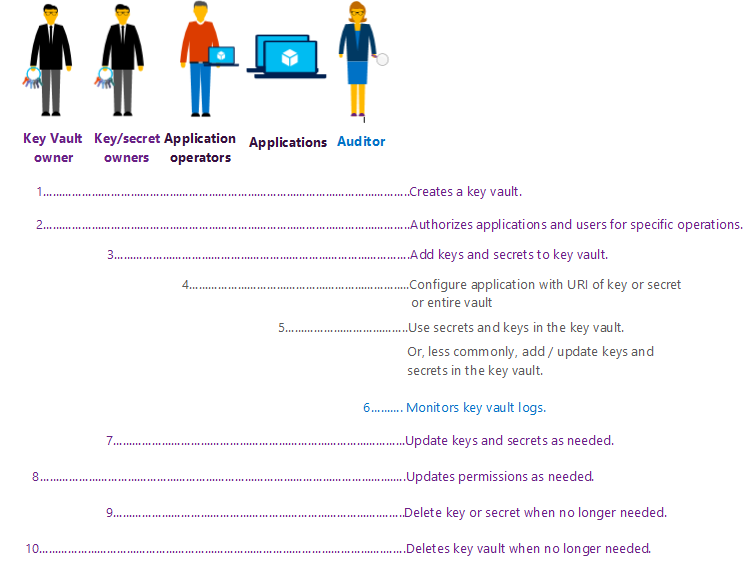
### Using Key Vault with ARM Template

### Encrypt and Decrypt BLOBS in Azure Storage

### **About Key Vault**

### Key Vault serves as a store of cryptographic keys and secrets, such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords.

* Developers can create keys for development and testing in minutes, and then seamlessly migrate them to production keys.
* Security administrators can grant (and revoke) permission to keys, as needed.



**Benefits:**

* Increase security and control over keys and passwords.
* Use FIPS 140-2 Level 2 validated hardware security modules (HSMs).
* Reduce latency with cloud scale and global redundancy.
* Create and import encryption keys in minutes.
* Applications have no direct access to keys.
* Simplify and automate tasks for SSL/TLS certificates.

**Secrets vs Keys**

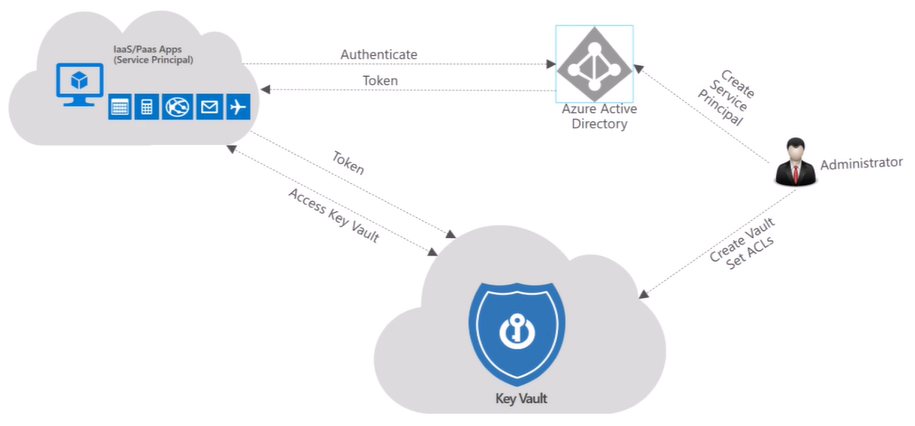
* A **secret** is essentially a small data blob (of up to 10 KB in size) that authorized users and applications **can retrieve from the vault**. To secure access to secrets, you create Azure Active Directory objects representing these users or applications, which they subsequently use to authenticate. Effectively, you avoid potential risk associated with users storing secrets in non-secure locations and eliminate the need to hard-code them into applications.
* Unlike secrets, **keys** stored in a vault **do not leave its boundaries**. Instead, once you add a key to the vault, users and applications must invoke cryptographic functions to perform any operations that require its knowledge. On the other hand, the ability to complete such invocation is also subject to a successful Azure Active Directory-based authentication.

**How to access keys and secrets:**

* To access keys and secrets, users and applications must possess valid Azure Active Directory tokens representing security principal with sufficient permissions to the target vault.
* You can use a REST-based API or Azure PowerShell to retrieve secrets and public parts of keys (in JSON format) from Key Vault.

**How it works:**

1. Applications that need access to Key Vault are registered with Azure Active Directory as Service Principals.
2. Administrator will then create a Key Vault and sets Access Control Lists on the vault so that applications can access it.
3. Applications then authenticates with Azure Active Directory and gets the Token.
4. The application then presents this token to Azure Key Vault
5. Azure Key vault then grants access based on Access Control List (ACL)



Administrator would sign in with an Azure subscription, create a vault for the organization in which to store keys, and then be responsible for operational tasks, such as:

* Create or import a key or secret.
* Revoke or delete a key or secret.
* Authorize users or applications to access the key vault, so they can then manage or use its keys and secrets.
* Configure key usage (for example, sign or encrypt).
* Monitor key usage.

This administrator would then provide developers with URIs to call from their applications, and provide their security administrator with key usage logging information.

**Walkthrough: Steps to create Hello Key Vault Application**

1. Create Service Principal (Azure AD Application)
2. Create Key Vaults and Set ACL's
3. Add Keys and Secrets to the Key Vault.
4. Get Sample Code and Edit config configuration.
5. Build and Run the application.

**Step 1: Create Azure Active Directory Application (Service Principal)**

1. Azure Portal 🡪 Azure Active Directory 🡪 App registrations 🡪 **+ New application registration**
2. Provide Name="KeyVaultDemoApp", Application type="Web app/API", Sign-on URL: <http://localhost/login> (anything is fine as it can be changed later)
3. KeyVaultDemoApp 🡪 Properties 🡪 **Copy App ID URI**
4. KeyVaultDemoApp 🡪 Certificates and Secrets 🡪 Click on **+ New client secret**
5. Provide Key description = K1, Select Duration, and then click **SAVE**. The page refreshes and now shows a key value. **Copy this Key**.

**Step 2: Create Key Vaults**

1. Azure Portal 🡪 All Services 🡪 Key Vault 🡪 +Add
2. Enter Name = DssDemoKeyVault
3. Select **Advanced** **access policy** to
   1. Enable access to Azure VM for deployment
   2. Enable access to Azure Resource Manager for template deployment
   3. Enable access to Azure Disk Encryption for volume encryption
4. Add Access Policy 🡪 +Add
5. Select Template = Key & Secret Management
6. Select Principals = "**KeyVaultDemoApp**" (AAD Application Created before)
7. Authorized Application is disabled

**OR**

1. Select Template = Key & Secret Management
2. Select Principals = **AD Users and Roles** who can access keys
3. Authorized Application = "KeyVaultDemoApp" (AAD Application Created before)
4. Create
5. Go to Created Key Vault 🡪 Essentials 🡪 Copy DNS Name: <https://dssdemokeyvault.vault.azure.net/>

**OR**

**PowerShell command to Create a key vault**

**New-AzKeyVault** -VaultName 'DssDemoKeyVault' -ResourceGroupName 'DemoRG' -Location 'East US'

**Set-AzKeyVaultAccessPolicy** -VaultName 'DssDemoKeyVault' -ServicePrincipalName 8f8c4bbd-485b-45fd-98f7-ec6300b7b4ed -PermissionsToKeys decrypt, sign

**Step 3: Add a key or secret to the key vault**

1. Select the DssDemoKeyVault 🡪 Keys +Add
2. Options = Generate, Name=MyFirstKey 🡪 Create
3. Click on Key 🡪 Current Version 🡪 Copy Identifier

OR

$key = ***Add-AzureKeyVaultKey*** -VaultName 'DssDemoKeyVault' -Name 'MyFirstKey' -Destination 'Software'

**Note**: To create an HSM-protected key, set the **-Destination** parameter to 'HSM'

You can reference a key that you created or uploaded to Azure Key Vault by using its URI.

1. Get the current version, you can use <https://dssdemokeyvalut.vault.azure.net/keys/MyFirstKey> and use [<https://dssdemokeyvalut.vault.azure.net/keys/MyFirstKey>/cgacf4f763ar42ffb0a1gca546aygd87](https://contosokeyvault.vault.azure.net/keys/ContosoFirstKey/cgacf4f763ar42ffb0a1gca546aygd87) to get this specific version.

**Step 4: `**

1. Create a New ASP.NET Web Application / ASP.NET Core Web Application
2. Add Nuget Packages
   1. Azure.Identity
   2. Azure.Security.KeyVault.Secrets

using Azure.Identity;

using Azure.Security.KeyVault.Secrets;

string clientId = "4cfd356c-08a8-4835-b0cd-e662787e2164";

string tenantID = "82d8af3b-d3f9-465c-b724-0fb186cc28c7";

string clientSecret = "aoG8Q~p~yRqcNSogn0D5oVeBpbO5jfC9lC\_Ukavh";

string keyVaultUrl = "https://dssdemo-keyvault.vault.azure.net";

string secretName = "S1";

var credential = new ClientSecretCredential(tenantID, clientId, clientSecret);

var client = new SecretClient(new Uri(keyVaultUrl), credential);

// Retrieve the secret

KeyVaultSecret sec = client.GetSecret(secretName);

// Print the secret value to the console

Console.WriteLine(sec.Value);

**Managed Identity**

**System Assigned Identity**: A system assigned managed identity enables Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based-access-control. The lifecycle of this type of managed identity is tied to the lifecycle of this resource. Additionally, each resource (e.g. Virtual Machine) can only have one system assigned managed identity.

**User Assigned:** User assigned managed identities enable Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. This type of managed identities are **created as standalone Azure resources**, and have their own lifecycle. A single resource (e.g. Virtual Machine) can utilize multiple user assigned managed identities. Similarly, a single user assigned managed identity can be shared across multiple resources (e.g. Virtual Machine).

**HomeController.cs**

public IActionResult Index()

{

string keyVaultUrl = "https://dssdemo-keyvault.vault.azure.net";

string secretName = "S1";

//Use the below line for System Assigned Identity

var client = new SecretClient(new Uri(keyVaultUrl), new DefaultAzureCredential());

//Use the below line for User Assigned Managed Identity along with its App-ID

var client = new SecretClient(new Uri(keyVaultUrl), new ManagedIdentityCredential("317a0f74-9382-45ec-bd0a-0cef1bf6993d"));

KeyVaultSecret sec = client.GetSecret(secretName);

ViewBag.Name = sec.Value;

return View();

}

**Encrypt and decrypt blobs in Microsoft Azure Storage using Azure Key Vault**

1. Azure Portal 🡪 Create a Storage Account with default settings
2. Visual Studio 🡪 Create a Console Based Application
3. Add the Following Nuget Packages
   1. Microsoft.IdentityModel.Clients.ActiveDirectory
   2. Microsoft.Azure.KeyVault
   3. Microsoft.Azure.KeyVault.Extensions
   4. Microsoft.Azure.Storage.Blob
4. Add the following to App.Config

<appSettings>

<add key="accountName" value="<Storage Account Name> "/>

<add key="accountKey" value="<Storage Account key>"/>

<add key="container" value="<containerName>"/>

<add key="clientId" value="<AADAppClientid>"/>

<add key="clientSecret" value="<AADAppClientSecret>"/>

</appSettings>

1. Add a new **class Utils** and the method to get an Access Token. This method is used by Key Vault classes that need to authenticate for access to your key vault.

public class Utils

{

//this is an optional property to hold the secret after it is retrieved

public static string EncryptSecret { get; set; }

//the method that will be provided to the KeyVaultClient

public static async Task<string> GetToken(string authority, string resource, string scope)

{

var authContext = new AuthenticationContext(authority);

ClientCredential clientCred = new ClientCredential(ConfigurationManager.AppSettings["ClientId"],

ConfigurationManager.AppSettings["ClientSecret"]);

AuthenticationResult result = await authContext.AcquireTokenAsync(resource, clientCred);

if (result == null)

throw new InvalidOperationException("Failed to obtain the JWT token");

return result.AccessToken;

}

}

1. Create a Console Application and In the Main function, add the following code

StorageCredentials creds = new StorageCredentials(ConfigurationManager.AppSettings["accountName"], ConfigurationManager.AppSettings["accountKey"]);

CloudStorageAccount account = new CloudStorageAccount(creds, useHttps: true);

CloudBlobClient client = account.CreateCloudBlobClient();

CloudBlobContainer container = client.GetContainerReference(ConfigurationManager.AppSettings["container"]);

container.CreateIfNotExists();

// The Resolver object is used to interact with Key Vault for Azure Storage.

// This is where the GetToken method from above is used.

KeyVaultKeyResolver cloudResolver = new KeyVaultKeyResolver(Utils.GetToken);

// Retrieve the key that you created previously.

// The IKey that is returned here is an RsaKey.

var rsa = cloudResolver.ResolveKeyAsync("https://dsskeyvault123.vault.azure.net/keys/MyFirstKey", CancellationToken.None).GetAwaiter().GetResult();

// Now you simply use the RSA key to encrypt by setting it in the BlobEncryptionPolicy.

BlobEncryptionPolicy policy = new BlobEncryptionPolicy(rsa, null);

BlobRequestOptions options = new BlobRequestOptions() { EncryptionPolicy = policy };

// Reference a block blob.

CloudBlockBlob blob = container.GetBlockBlobReference("MyFile.txt");

// Upload using the UploadFromStream method.

using (var stream = System.IO.File.OpenRead(@"D:\Demo.txt"))

blob.UploadFromStream(stream, stream.Length, null, options, null);

//In this case, we will not pass a key and only pass the resolver because this policy will only be used for downloading / decrypting.

policy = new BlobEncryptionPolicy(null, cloudResolver);

options = new BlobRequestOptions() { EncryptionPolicy = policy };

using (var np = File.Open(@"d:\MyFileDecrypted.txt", FileMode.Create))

blob.DownloadToStream(np, null, options, null);

**Make Sure the AD App has enough permission to Unwrap – otherwise Exception will be thrown**

Key Vault 🡪 Access policies 🡪 For AD App 🡪 Expand Key Permissions 🡪 Check Unwrap

1. Note the file in container is encrypted but with access to key we are able to fetch the same.