KEYFACTOR

Installation and Reference Guide
Keyfactor Secrets Engine Plugin for Hashicorp Vault

Contents

Overview

Keyfactor enables DevOps teams to get seamless access to trusted internal and public certificates via native Vault API calls and commands, while security teams maintain complete visibility and control over backend PKI operations.

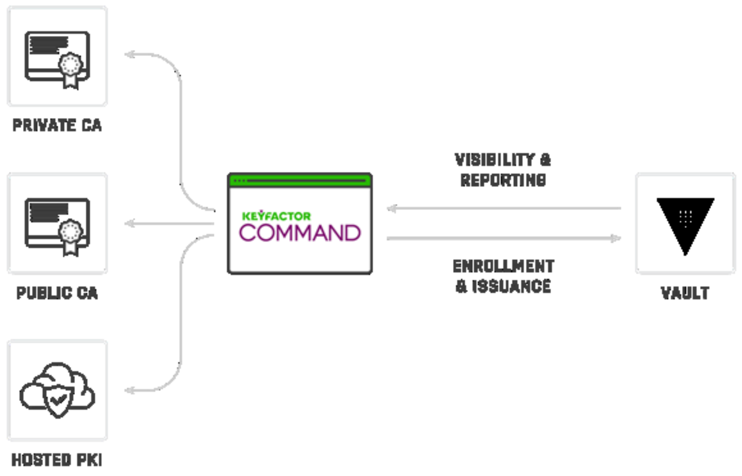
The Keyfactor Secrets Engine provides a PKI backend for Vault to issue trusted certificates via the Keyfactor platform.

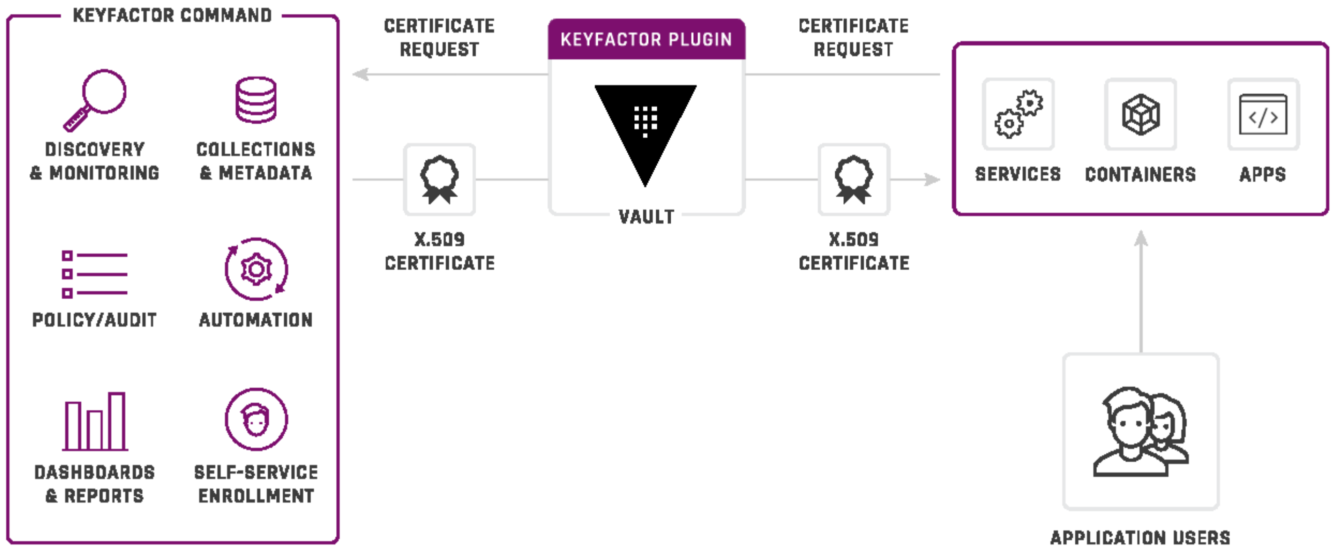
Enables developers to use native Vault API calls and commands to request certificates from Keyfactor

Allows security teams to maintain visibility and control over all certificates issued to Vault instances

Connects Vault with trusted public, private, or cloud-hosted CAs configured in the Keyfactor platform.

Certificates are requested through Vault using standard Vault commands, and are then redirected to Keyfactor so that the certificates can be issued off of a trusted enterprise certificate authority.   After issuance, the certificate is then returned to Hashicorp Vault and stored within the Vault Secrets store to then be used by other application.





High-level Design and Benefits

The Keyfactor Secrets Engine for Hashicorp Vault is a Vault plugin that replicates Vault's onboard PKI API and processes certificate enrollment requests through the Keyfactor Command or Keyfactor Control platform. In many cases the onboard PKI engine included with Vault can be swapped for the Keyfactor engine seamlessly, with no impact to Vault client applications. While the simplicity of the onboard PKI is attractive to developers who are trying to implement the simplest solution in order to meet encryption requirements, it presents other enterprise teams with some challenges when it comes to PKI operations and security:

- The Vault infrastructure and root materials are not managed by PKI professionals and policies, but rather by DevOps teams that may not be trained in how to properly handle and manage an enterprise PKI.
- Lack of Certificate Lifecycle Management places organizations in a reactionary posture. If there are weaknesses in the organization processes, full visibility of the certificates is necessary in order to identify these risks prior to a security event or audit failure.
- All certificates are susceptible as an attack surface and should be managed and monitored, regardless of their lifetime, to ensure that issuance policies and certificate standards are followed.

Keyfactor Command can provide the control and visibility needed for a Vault environment. Using the Keyfactor Secrets Engine plugin for Vault, PKI functionality is directed to your enterprise PKI environment, placing control back into the hands of the enterprise PKI admins, and allowing your PKI admins to stay in control of how and when certificates are issued. The Keyfactor Secrets Engine offers the following enterprise capabilities:

- Issue certificates and place them into the Vault secrets store using your existing enterprise PKI.
- Eliminate the need for a standalone PKI within the vault environment.
- Gain complete visibility and management of certificates across all Vault instances and manage them through a single pane of glass.
- Reporting, alerting, automation, and auditing on the certificates within the environment.
- Easily identify and revoke non-compliant or rogue certificates.
- Integrate with SIEMs and ticketing systems for automated notifications.

Quickstart

1. Install Vault and Keyfactor in the target environment, ensuring the Vault server has connectivity to Keyfactor.
2. Create an API application in Keyfactor to be used by the plugin.
3. Create or identify a user in a Security Role with the following permissions:
    a. API: Read
    b. Certificate Enrollment: Enroll CSR
    c. Certificates: Revoke (optional)
4. Copy the Keyfactor Secrets Engine binary into your Hashicorp Vault plugins directory
5. Create a JSON file on the machine hosting the engine with the following parameters:

Parameter	Value
host	Hostname or IP address of Keyfactor server.
creds	Basic auth credentials for the user identified in step 3 (base-64 encoding of “DOMAIN\user:Password”).
appkey	Base-64 encoding of the API application key.
secret	Base-64 encoding of the API secret key.
template	Active Directory certificate template to use for certificate requests.
protocol	One of “http” or “https”. For making requests to the Keyfactor server.
CA	Hostname\\\\LogicalName

```
Example:
{
        "host":"192.168.0.24",
        "creds":"SkRLXGpraWxnYWxsOlBAc3N3MHJk",
        "appkey":" LBfep9KOH0sNHg==",
        "secret": "AbIDf2NUNh41oQ==",
        "template":"User",
        "protocol":"https",
        "CA":"CA1.jdk.cms\\\\jdk-CA1"
}
}
```

(Note: Ensure permissions are set appropriately on this file to avoid credential disclosure)

6. Point the plugin to the configuration file with the KF_CONF_PATH environment variable
    export KF_CONF_PATH=/path/to/json/file

7. Enable the Keyfactor Secrets Engine plugin in your Vault instance
    - `vault write sys/plugins/catalog/keyfactor sha256=47f549d44ab2abcb528aa45725b3a83334a9465bb487f3d1182add55e5580c36 command="keyfactor"`
      -Depending on version of Vault, “sys/plugins/catalog/keyfactor” may need to be replaced with “sys/plugins/catalog/secret/keyfactor”
    - `vault secrets enable keyfactor`

Requirements

The requirements for the plugin are relatively simple.   It runs as a single executable on the Hashicorp Vault server. There are no specific system requirements to install it, however there are a few general things that must be in place for it to function properly. These requirements are listed below, and are then expanded in the details throughout this document.

1. General Keyfactor Requirements

- A functional instance of Keyfactor Command
- An administrative user account to be used for configuring the Keyfactor options needed for the implementation
- A functional integrated certificate authority to be used for issuing the certificates
- A certificate template (or templates) defined to use for certificate issuance.
- A user account with permissions to connect to the Keyfactor API and submit certificate requests. This user account will require READ and ENROLL permissions on the certificate template that you will use for the Vault plugin.
- An API key within the Keyfactor systemthat will be used to authenticate the enrollment process from the Hashicorp Vault server.

2. General Hashicorp Vault Requirements

- A functional Hashicorp Vault Installation
- An administrative account with permission to login to the Hashicorp Vault server in order to make administrative changes.
- An adequate number of unseal keys to meet the minimum criteria to unseal the Hashicorp Vault
- A Hashicorp Vault login token

Setup Procedures – Keyfactor

3. Create the Active Directory service account

For the purposes of this document, we will not go into the details of how to create an Active Directory user since this process varies widely by company, however, here are a couple things to consider:

- Ensure that the user does not have an expiring password, or if it does, ensure that the password resets are managed carefully.   Expiration of this password could result in production outages with the plugin.
- Ensure that the user does not have logon time restrictions unless you only want the Hashicorp Vault plugin to function during specific timeframes.

4. Assign the user permissions in Keyfactor Command

In order to be able to enroll for certificates through the Keyfactor Command API, it will be necessary to create the necessary role and delegate permissions within Keyfactor.   It is not a requirement that this be a new role. If there is an existing role within your organization that allows for these basic permissions, that role can be used for this connection. If you do not have an existing role, and would like to create one, those steps are provided below.

5. Create the certificate template to use

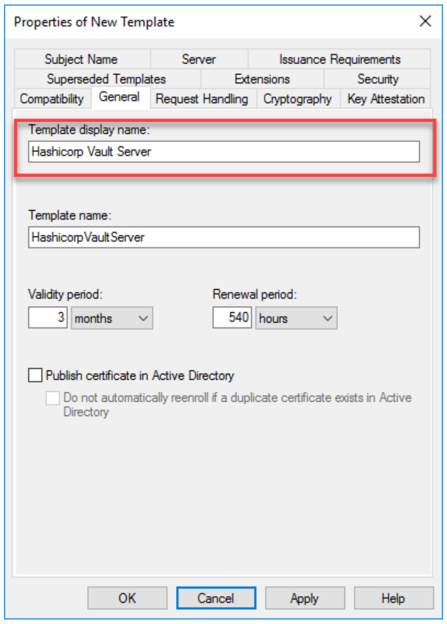
The first step to configuring Keyfactor is to create the certificate template that will be used for the enrollment and publish it into Keyfactor.

To do this:

1. Open up the Certificate Authority MMC console.
2. Right Click on Certificate Templates, and select “Manage”. This will open up the Certifcate Templates MMC console.
3. In the Certificate Templates MMC console, choose a template that you would like to use as a starting point for your new Vault Plugin template, and duplicate it as a starting point.   For standard SSL certificates, most

companies will start with a template such as "Web Server" for a general template. In situations where you need the certificate to do mutual TLS authentication, you may wish to choose the Computer template so that it will include both the Client Authentication and Server Authentication key usages. To duplicate the template, right click on the template and select "Duplicate Template"

4. You should now see the properties for the new template you are creating, and you will need to customize the template for use with the plugin. In most cases, there will be only a few minor changes that need made to the template.
   a. On the General tab, change the Template Display Name to represent the name that you want to have on the template.



Template display name: Hashicorp Vault Server

Template name: HashicorpVaultServer

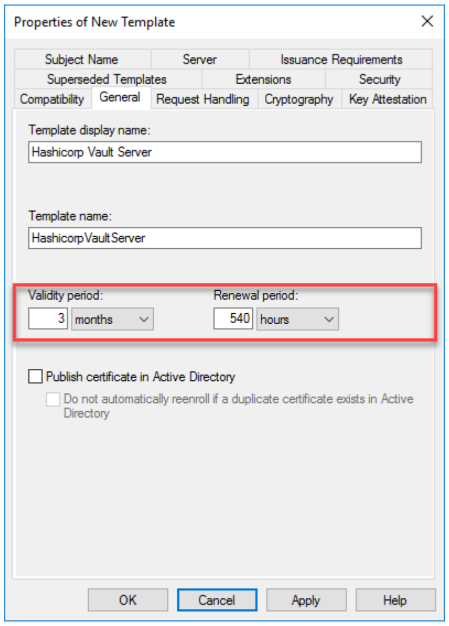
Validity period: 3 months

Renewal period: 540 hours

☐ Publish certificate in Active Directory

☐ Do not automatically reenroll if a duplicate certificate exists in Active Directory

   b. On the General tab, set the Validity Period and Renewal Period for the certificates that you are going to issue off of this template. This setting will vary by company policy, and use cases.



Template display name: Hashicorp Vault Server

Template name: HashicorpVaultServer

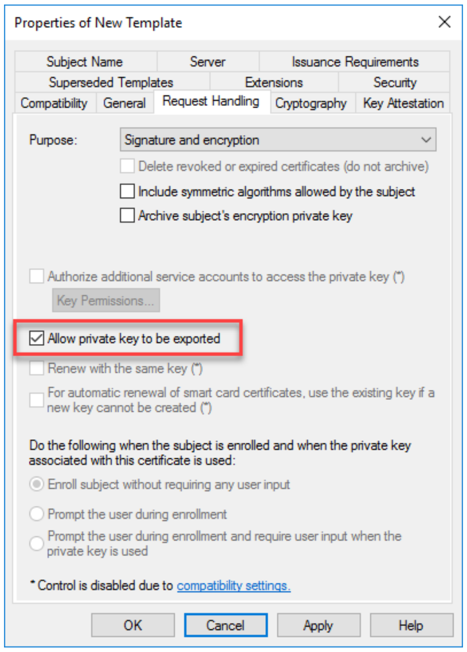
Validity period: 3 months

Renewal period: 540 hours

☐ Publish certificate in Active Directory

☐ Do not automatically reenroll if a duplicate certificate exists in Active Directory

   c. On the Request Handling tab, ensure that the option is selected for "Allow Private Keys to be Exported"



Purpose: Signature and encryption

[ ] Delete revoked or expired certificates (do not archive)

[ ] Include symmetric algorithms allowed by the subject

[ ] Archive subject's encryption private key

[ ] Authorize additional service accounts to access the private key (*)

[✓] Allow private key to be exported

[ ] Renew with the same key (*)

[ ] For automatic renewal of smart card certificates, use the existing key if a new key cannot be created (*)

Do the following when the subject is enrolled and when the private key associated with this certificate is used:

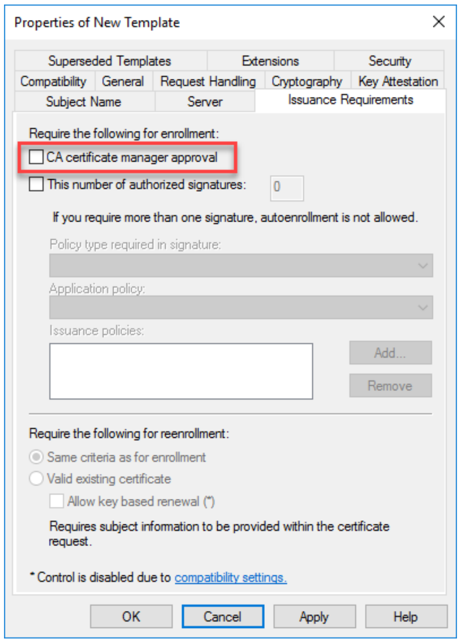
(●) Enroll subject without requiring any user input

( ) Prompt the user during enrollment

( ) Prompt the user during enrollment and require user input when the private key is used

* Control is disabled due to compatibility settings.

d. Unless you are planning to implement an approval workflow process for the certificates issued through Hashicorp Vault, ensure that "CA Certificate Manager Approval" is not checked on the Issuance Requirements tab.



Require the following for enrollment:

[ ] CA certificate manager approval

[ ] This number of authorized signatures: 0

If you require more than one signature, autoenrollment is not allowed.

Policy type required in signature:

Application policy:

Issuance policies:

Require the following for reenrollment:

(●) Same criteria as for enrollment

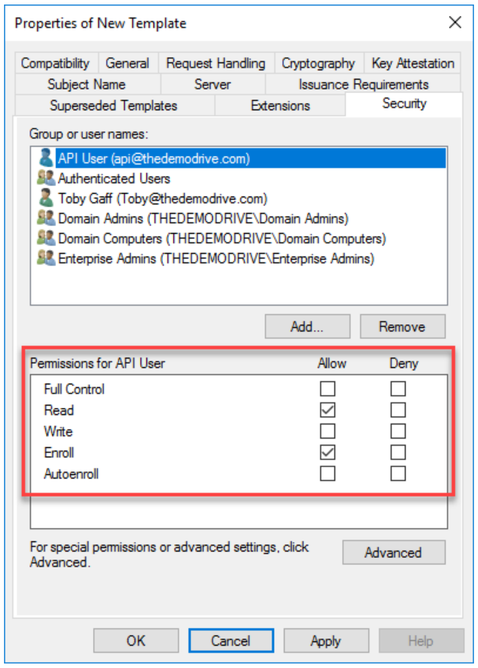
( ) Valid existing certificate

[ ] Allow key based renewal (*)

Requires subject information to be provided within the certificate request.

* Control is disabled due to compatibility settings.

e. On the Security tab, add the service account that was created earlier so that it has permissions to enroll certificates off of this template. Click Add to search for the user to add, and then grant the user READ and ENROLL permissions on the Template.


Properties of New Template

Compatibility | General | Request Handling | Cryptography | Key Attestation | Subject Name | Server | Issuance Requirements | Superseded Templates | Extensions | Security

Group or user names:

- API User (api@thedemodrive.com)
- Authenticated Users
- Toby Gaff (Toby@thedemodrive.com)
- Domain Admins (THEDEMODRIVE\Domain Admins)
- Domain Computers (THEDEMODRIVE\Domain Computers)
- Enterprise Admins (THEDEMODRIVE\Enterprise Admins)

Permissions for API User	Allow	Deny
Full Control	☐	☐
Read	✓	☐
Write	☐	☐
Enroll	✓	☐
Autoenroll	☐	☐

For special permissions or advanced settings, click Advanced.

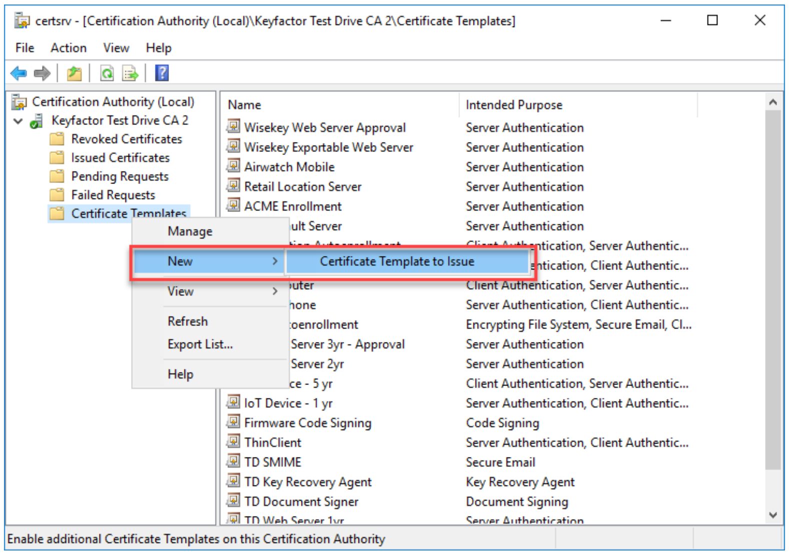
f. Click OK to save the template.

6. Publish the template on the Certificate Authority

It is now necessary to take the certificate template that you created and publish it so that it is an available template for issuance off of the CA.

To publish the template:

1. Open the Certificate Authority MMC console
2. Right on Certificate Templates in the left hand pane and select NEW – Certificate Template to Issue


certsrv - [Certification Authority (Local)\Keyfactor Test Drive CA 2\Certificate Templates]

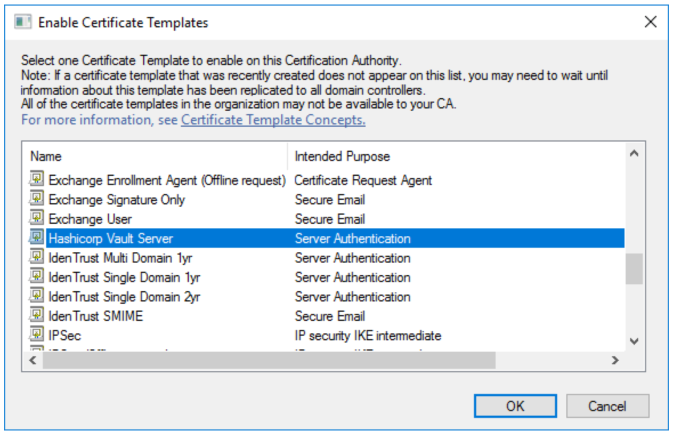
Certification Authority (Local) > Keyfactor Test Drive CA 2: Revoked Certificates, Issued Certificates, Pending Requests, Failed Requests, Certificate Templates

Manage / New > Certificate Template to Issue / View / Refresh / Export List... / Help

Name	Intended Purpose
Wisekey Web Server Approval	Server Authentication
Wisekey Exportable Web Server	Server Authentication
Airwatch Mobile	Server Authentication
Retail Location Server	Server Authentication
ACME Enrollment	Server Authentication
ult Server	Server Authentication
	Client Authentication, Server Authentic...
	ntication, Client Authentic...
uter	Client Authentication, Server Authentic...
hone	Server Authentication, Client Authentic...
oenrollment	Encrypting File System, Secure Email, Cl...
Server 3yr - Approval	Server Authentication
Server 2yr	Server Authentication
ce - 5 yr	Client Authentication, Server Authentic...
IoT Device - 1 yr	Server Authentication, Client Authentic...
Firmware Code Signing	Code Signing
ThinClient	Server Authentication, Client Authentic...
TD SMIME	Secure Email
TD Key Recovery Agent	Key Recovery Agent
TD Document Signer	Document Signing
TD Web Server 1yr	Server Authentication

Enable additional Certificate Templates on this Certification Authority

3. Select the template that was created in the previous step, and then click OK.


Enable Certificate Templates

Select one Certificate Template to enable on this Certification Authority.
Note: If a certificate template that was recently created does not appear on this list, you may need to wait until information about this template has been replicated to all domain controllers.
All of the certificate templates in the organization may not be available to your CA.
For more information, see Certificate Template Concepts.

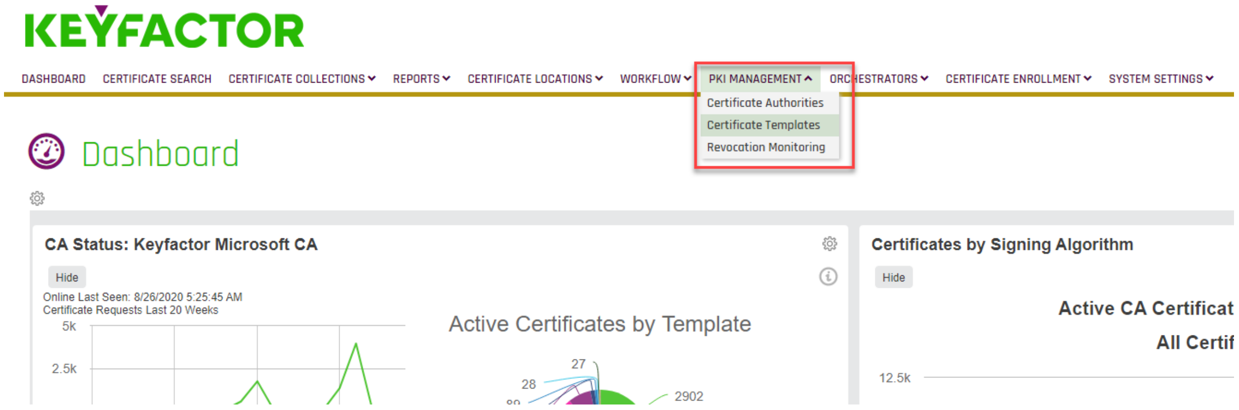
Name	Intended Purpose
Exchange Enrollment Agent (Offline request)	Certificate Request Agent
Exchange Signature Only	Secure Email
Exchange User	Secure Email
Hashicorp Vault Server	Server Authentication
IdenTrust Multi Domain 1yr	Server Authentication
IdenTrust Single Domain 1yr	Server Authentication
IdenTrust Single Domain 2yr	Server Authentication
IdenTrust SMIME	Secure Email
IPSec	IP security IKE intermediate

7. Import the new template into the Keyfactor console

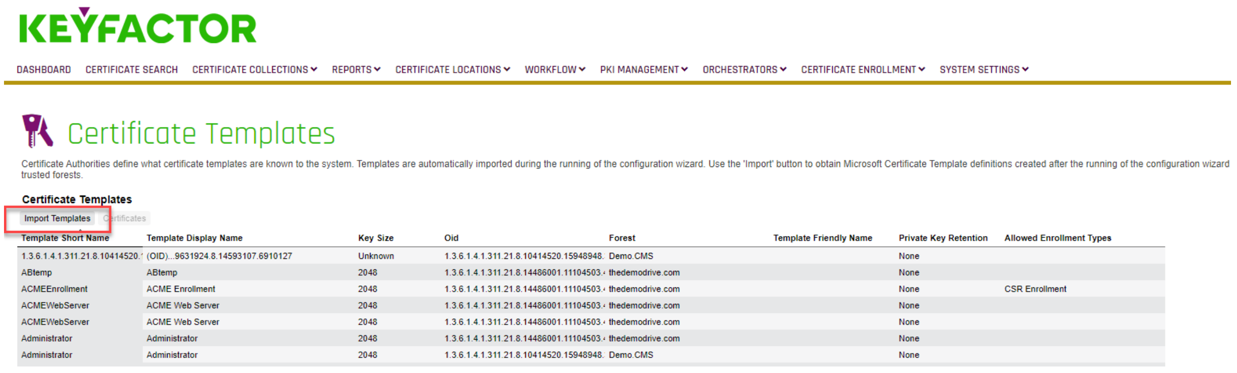
Now that the new certificate template has been created on the CA, we need to ensure that the template is available for issuance within the Keyfactor Command console.

To import the certificate template:

1. Log into the Keyfactor Command console as a user with administrative privileges
2. Select PKI Management from the top menu bar, then select “Certificate Templates”



3. Click the Import Templates button at the top of the screen.



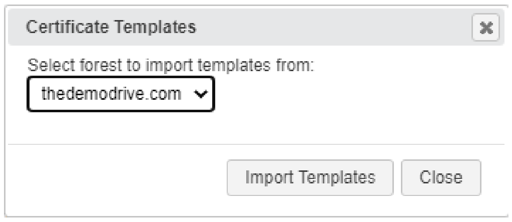
Certificate Templates

Certificate Authorities define what certificate templates are known to the system. Templates are automatically imported during the running of the configuration wizard. Use the 'Import' button to obtain Microsoft Certificate Template definitions created after the running of the configuration wizard trusted forests.

Certificate Templates

Template Short Name	Template Display Name	Key Size	Oid	Forest	Template Friendly Name	Private Key Retention	Allowed Enrollment Types
1.3.6.1.4.1.311.21.8.10414520.1	(OID)...9631924.8.14593107.6910127	Unknown	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	
ABtemp	ABtemp	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
ACMEEnrollment	ACME Enrollment	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	CSR Enrollment
ACMEWebServer	ACME Web Server	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
ACMEWebServer	ACME Web Server	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
Administrator	Administrator	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
Administrator	Administrator	2048	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	

4. Select the Active Directory Forest where you created the template, and then click “Import Templates”


Certificate Templates

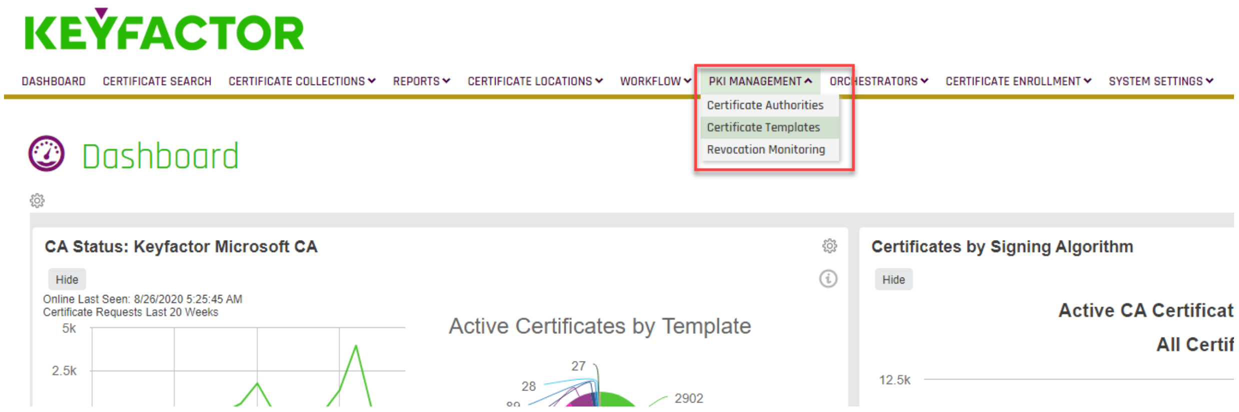
Select forest to import templates from: thedemodrive.com

8. Enable the template for CSR enrollment through Keyfactor Command

Once the template has been imported into Keyfactor Command, it is then necessary to enable that template for PFX enrollment through the console.

To enable CSR enrollment on the template:

1. Select PKI Management from the top menu bar, then select “Certificate Templates”

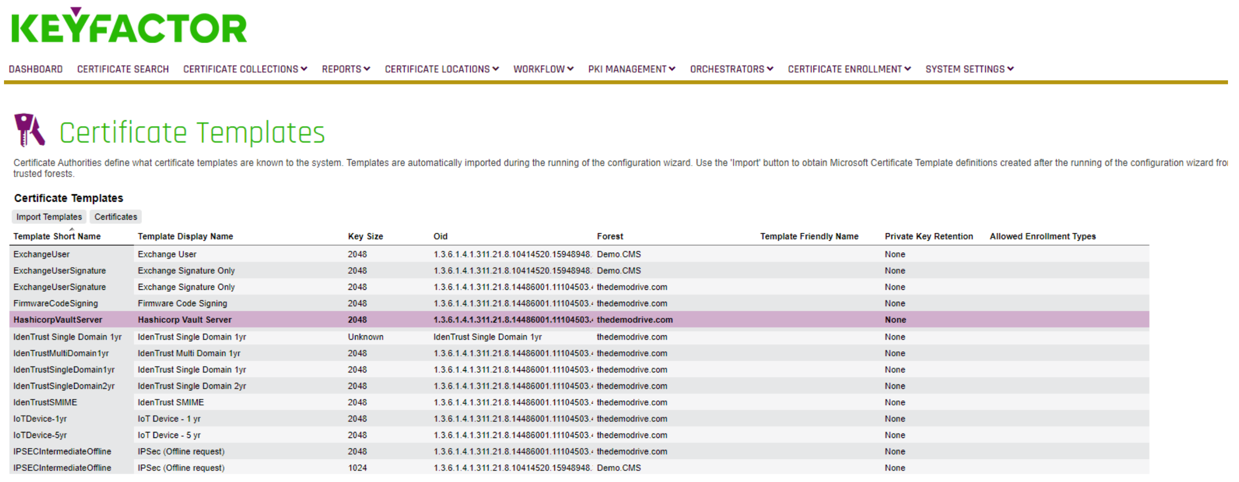


CA Status: Keyfactor Microsoft CA

Online Last Seen: 8/26/2020 5:25:45 AM
Certificate Requests Last 20 Weeks

Certificates by Signing Algorithm

2. Look through the list to locate the certificate template that was created and previously imported into the Keyfactor console. Double Click on that template.



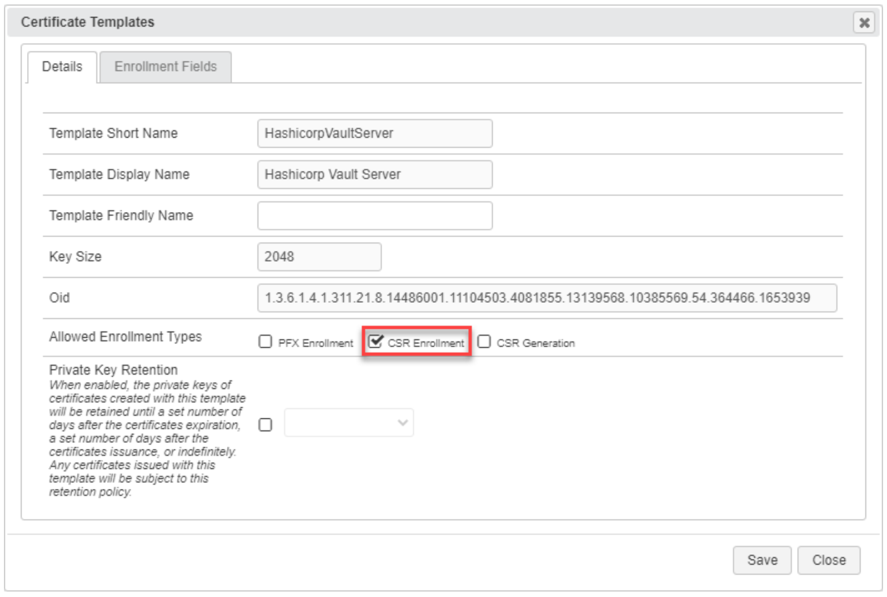
Certificate Templates

Certificate Authorities define what certificate templates are known to the system. Templates are automatically imported during the running of the configuration wizard. Use the 'Import' button to obtain Microsoft Certificate Template definitions created after the running of the configuration wizard fro trusted forests.

Certificate Templates

Template Short Name	Template Display Name	Key Size	Oid	Forest	Template Friendly Name	Private Key Retention	Allowed Enrollment Types
ExchangeUser	Exchange User	2048	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	
ExchangeUserSignature	Exchange Signature Only	2048	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	
ExchangeUserSignature	Exchange Signature Only	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
FirmwareCodeSigning	Firmware Code Signing	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
HashicorpVaultServer	**Hashicorp Vault Server**	**2048**	**1.3.6.1.4.1.311.21.8.14486001.11104503.**	**thedemodrive.com**		**None**	
IdenTrust Single Domain 1yr	IdenTrust Single Domain 1yr	Unknown	IdenTrust Single Domain 1yr	thedemodrive.com		None	
IdenTrustMultiDomain1yr	IdenTrust Multi Domain 1yr	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IdenTrustSingleDomain1yr	IdenTrust Single Domain 1yr	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IdenTrustSingleDomain2yr	IdenTrust Single Domain 2yr	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IdenTrustSMIME	IdenTrust SMIME	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IoTDevice-1yr	IoT Device - 1 yr	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IoTDevice-5yr	IoT Device - 5 yr	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IPSECIntermediateOffline	IPSec (Offline request)	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
IPSECIntermediateOffline	IPSec (Offline request)	1024	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	

3. On the properties tab for the template, enable CSR enrollment. Then click OK



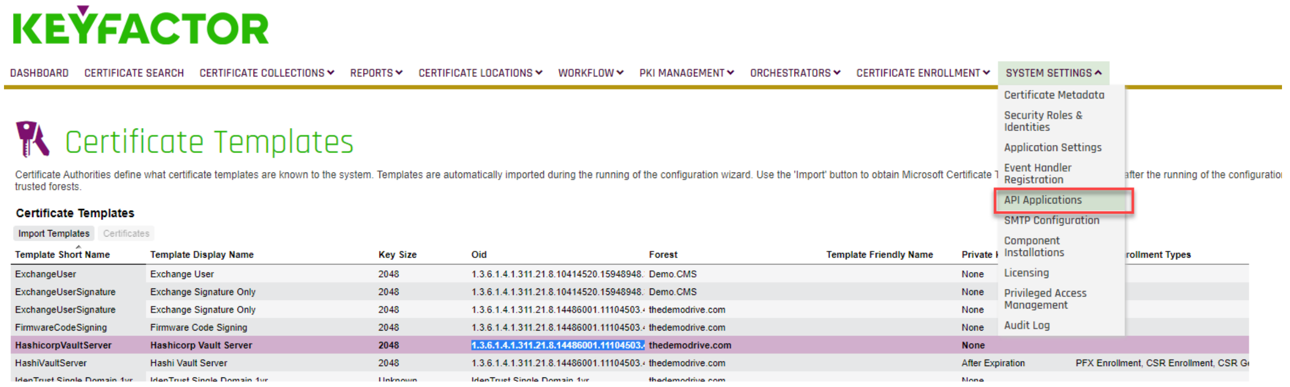
Template Short Name	HashicorpVaultServer
Template Display Name	Hashicorp Vault Server
Template Friendly Name	
Key Size	2048
Oid	1.3.6.1.4.1.311.21.8.14486001.11104503.4081855.13139568.10385569.54.364466.1653939
Allowed Enrollment Types	☐ PFX Enrollment [✓ CSR Enrollment] ☐ CSR Generation
Private Key Retention *When enabled, the private keys of certificates created with this template will be retained until a set number of days after the certificates expiration, a set number of days after the certificates issuance, or indefinitely. Any certificates issued with this template will be subject to this retention policy.*	☐

9. Create the Keyfactor API Key for the secrets engine

All enrollment processes that happen through the Keyfactor API require both basic authentication for the user account doing the enrollment, as well as an API key to authenticate the application connection and allowed templates for enrollment, therefore, we will need to setup an API key to be used by the Hashicorp Vault plugin.

To create a new API key and enable template access:

1. Go to the System Settings menu in Keyfactor Command. In Keyfactor 7 and previous versions, this is an option on the top menu bar of the application. In Keyfactor 8 or later, this has moved to a Gear icon in the top right corner of the screen. Select API Applications from the System Settings menu.



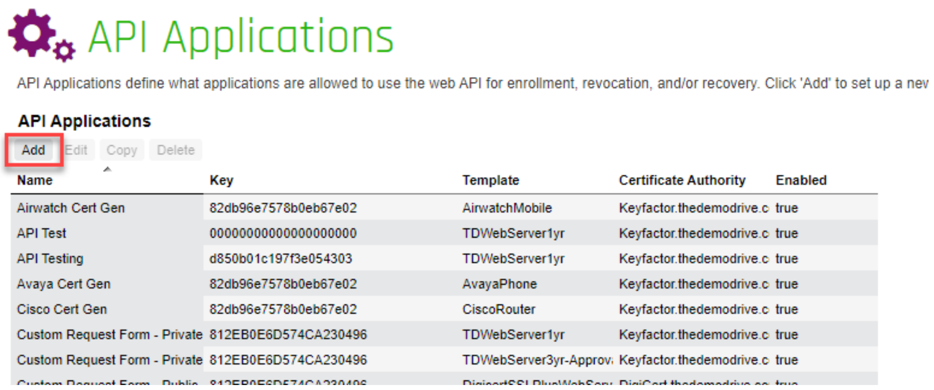
Certificate Templates

Certificate Authorities define what certificate templates are known to the system. Templates are automatically imported during the running of the configuration wizard. Use the 'Import' button to obtain Microsoft Certificate T after the running of the configuratio trusted forests.

Certificate Templates

Template Short Name	Template Display Name	Key Size	Oid	Forest	Template Friendly Name	Private	rollment Types
ExchangeUser	Exchange User	2048	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	
ExchangeUserSignature	Exchange Signature Only	2048	1.3.6.1.4.1.311.21.8.10414520.15948948.	Demo.CMS		None	
ExchangeUserSignature	Exchange Signature Only	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
FirmwareCodeSigning	Firmware Code Signing	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
HashicorpVaultServer	Hashicorp Vault Server	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		None	
HashiVaultServer	Hashi Vault Server	2048	1.3.6.1.4.1.311.21.8.14486001.11104503.	thedemodrive.com		After Expiration	PFX Enrollment, CSR Enrollment, CSR G

2. On the API Applications screen, select ADD.



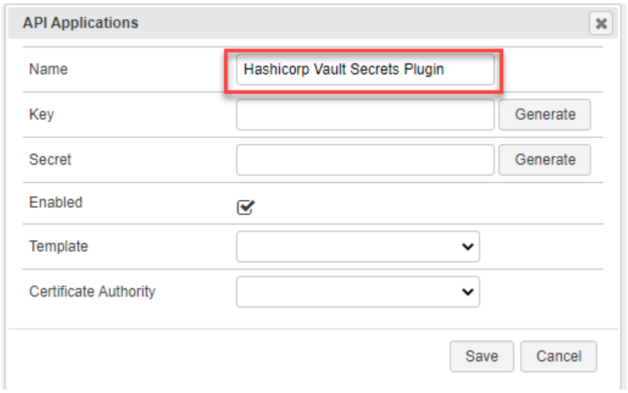
API Applications

API Applications define what applications are allowed to use the web API for enrollment, revocation, and/or recovery. Click 'Add' to set up a nev

API Applications

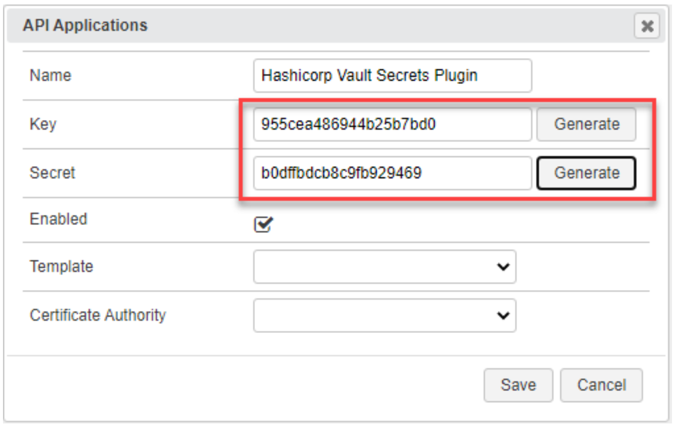
Name	Key	Template	Certificate Authority	Enabled
Airwatch Cert Gen	82db96e7578b0eb67e02	AirwatchMobile	Keyfactor.thedemodrive.c	true
API Test	00000000000000000000	TDWebServer1yr	Keyfactor.thedemodrive.c	true
API Testing	d850b01c197f3e054303	TDWebServer1yr	Keyfactor.thedemodrive.c	true
Avaya Cert Gen	82db96e7578b0eb67e02	AvayaPhone	Keyfactor.thedemodrive.c	true
Cisco Cert Gen	82db96e7578b0eb67e02	CiscoRouter	Keyfactor.thedemodrive.c	true
Custom Request Form - Private	812EB0E6D574CA230496	TDWebServer1yr	Keyfactor.thedemodrive.c	true
Custom Request Form - Private	812EB0E6D574CA230496	TDWebServer3yr-Approv	Keyfactor.thedemodrive.c	true

3. Enter a logical name for this API application. This value is arbitrary and can be anything that describes the connection.



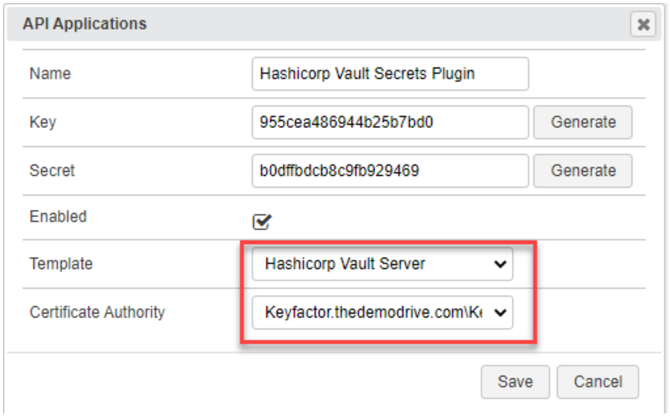
Name	Hashicorp Vault Secrets Plugin
Key	
Secret	
Enabled	✓
Template	
Certificate Authority	

4. You will need an API keys for the integration, this key can either be automatically generated by Keyfactor, or you can paste in a value from another key management platform. Both keys must be present in the configuration, however the Secret key is not used for this specific integration.


API Applications

Name	Hashicorp Vault Secrets Plugin
Key	955cea486944b25b7bd0
Secret	b0dffbdcb8c9fb929469
Enabled	✓
Template	
Certificate Authority	

5. Next, assign the template and Certificate Authority that you created earlier. This will control which template this API key is allowed to enroll off of. It is possible to configure multiple API applications with the same key to enable enrollment off of multiple templates.


API Applications

Name	Hashicorp Vault Secrets Plugin
Key	955cea486944b25b7bd0
Secret	b0dffbdcb8c9fb929469
Enabled	✓
Template	Hashicorp Vault Server
Certificate Authority	Keyfactor.thedemodrive.com\K

6. Click OK to save the information.

Setup Procedure – Hashicorp Vault Server Components

This document covers configuration of the Keyfactor Secrets Engine Plugin for Hashicorp Vault, and assumes that there is a running Vault environment in place. This document does not cover the steps necessary to do the initial install and configuration of Hashicorp Vault.

On the linux server where the plugin will reside, it will be necessary to setup the appropriate environment variables and configuration files to enable the plugin to run and establish a connection back to Keyfactor Command. Many of these commands and formats will vary based linux distribution and tools used. For this reason, some of these commands may need to be modified to work in your specific environment.

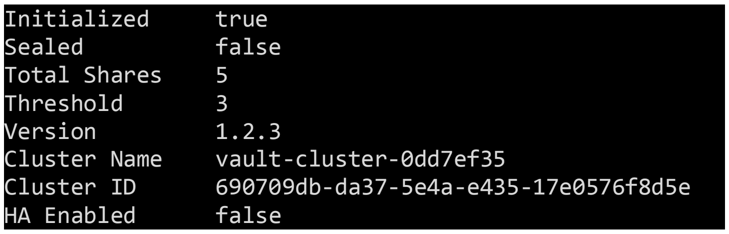
1. Confirm functionality of the Hashicorp Vault environment

   Hashicorp Vault must be installed and running to install and register the Keyfactor Secrets Plugin. To check the status of the Vault installation on the server being used, log into the Vault server with a logon account that has sufficient administrative privileges, and issue the following command:

```
vault status
```

   The results returned should look something like this:

```
Key             Value
---             -----
Seal Type       shamir
```



```
Initialized     true
Sealed          false
Total Shares    5
Threshold       3
Version         1.2.3
Cluster Name    vault-cluster-0dd7ef35
Cluster ID      690709db-da37-5e4a-e435-17e0576f8d5e
HA Enabled      false
```

There are 2 key values to look for in these results:

1. **Initialized** – Verify that the vault has been initialized and the status shows TRUE. If this value is FALSE, it means that this is a new instance of Vault that has not yet been initialed. You will need to go through a Vault initialization using the “vault operator init” command prior to proceeding.

2. **Unsealed** – In order to perform operations against a vault instance, the vault must be unsealed with the unlock keys that were generated during the initialization process. Vault has a (m of n) key requirement… so for instance 3 of the 5 keys must be entered to unseal the vault. (This requirement can vary based on configuration) If the current status of the vault shows SEALED then it is necessary to enter these keys to unseal the vault. These keys can be entered individually (and potentially by different people or processes)

```
vault operator unseal 3TGWPmQDdqkRsV9VamEYJ0tolsaEEo3u4kuwy2o6u6Om
vault operator unseal Ja4AGQ4N8193/5O7hJPpRcncmqBpnH1mdjqcQSqDVq6v
vault operator unseal cuE1X01NrNgeAU6ao5aNUFsjWAPhOgEPkgaW5Vl19XDg
```

… or they can be all issued within a single command as illustrated below

```
vault operator unseal 3TGWPmQDdqkRsV9VamEYJ0tolsaEEo3u4kuwy2o6u6Om && vault operator unseal
   Ja4AGQ4N8193/5O7hJPpRcncmqBpnH1mdjqcQSqDVq6v && vault operator unseal
   cuE1X01NrNgeAU6ao5aNUFsjWAPhOgEPkgaW5Vl19XDg
```

Once the appropriate number of keys has been entered, the status should indicate “Unsealed = True”

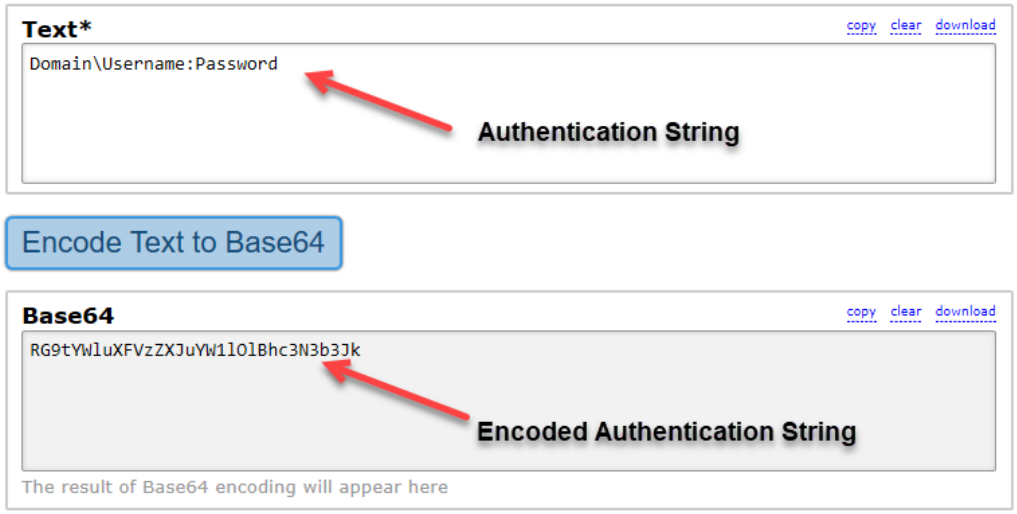
2. Encode the user credentials

In order to connect to the Keyfactor system, the Keyfactor plugin will need to use the credentials for the Active Directory Service Account that you created earlier in the process. These credentials will be used within Keyfactor plugin config file, and must be in Base64 encoded Basic Auth format.

To create the encoded string to use in the config file, use a Text to Base64 converter such as the one provided here:

https://base64.guru/converter/encode/text

The credential string to be encoded should be in the format of: Domain\Username:Password



```
Domain\Username:Password
```

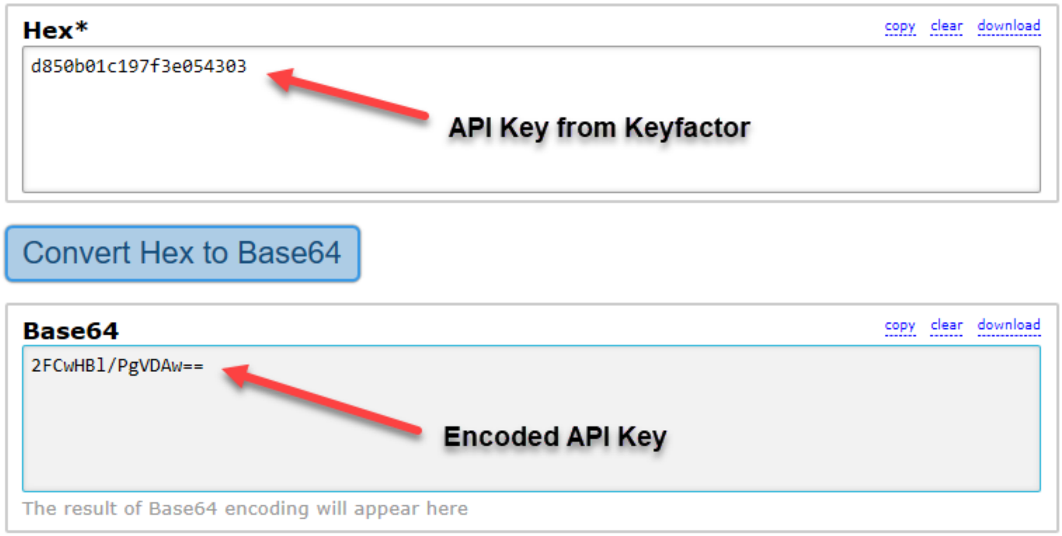
```
RG9tYWluXFVzZXJuYW1lOlBhc3N3b3Jk
```

You will need this encoded string in the following steps.

3. Encode the API Key

The API key that will be used to connect to Keyfactor will also need to be encoded. This, however, is a different encoding than what was used to encode the user string. This is due to the fact that the API key is actually a HEX value, and not just ASCII. In order to properly encode the API key, this must be done using a HEX to Base64 encoding tool like the one available here:

https://base64.guru/converter/encode/hex



```
d850b01c197f3e054303
```

```
2FCwHBl/PgVDAw==
```

You will need this encoded API key when building the config file for the plugin.

4. Build the Keyfactor Config file

The Keyfactor plugin requires a configuration file to be built to store the parameters for the connection to Keyfactor. This file is in a standard JSON format on the Hashicorp server. The file can be stored anywhere, but is typically in a location such as: /usr/bin/keyfactor/config.json

Based on the values below, create the config file and save it into a location on the Vault server.

The file will look something like this:

```
{
        "host":"192.168.160.60",
        "creds":"dGhlZGVtb2RyaXZlXHN2Y19rZXlmYWN0b3I6Y203SHRSTUVLelpuWEtGS3FpQks=",
1.          "appkey":"E23tz41aYzZj/A==",
        "template":"HashicorpVaultServer",
        "protocol":"https",
        "CA":"keyfactor.thedemodrive.com\\\\Keyfactor Test Drive CA 2"
}
```

The values within the file are as follows:

HOST – This is is the hostname or IP Address for the Keyfactor server.

CREDS – This is the username and password for the service account used to connect to Keyfactor encoded as a Base64 Basic Auth string. The details for creating this are in section 11 above.

APPKEY – This is the API key that was created in the Keyfactor system and Base64 encoded. As per the notes in section 12, this must be HEX to Base64 encoded.

TEMPLATE – The Template value is the Template Short Name for the template that was created to be used for the certificate issuance. This template nme must be the same template that is setup on the API key within Keyfactor, and the user account specified in the CREDS parameter must have READ and ENROLL permissions on this template.

PROTOCOL – The protocol used to connect to the Keyfactor server. This can be HTTP or HTTPS. Keyfactor recommends that HTTP only be used for sandbox or testing scenarios.

CA – This is the CA name for the certificate authority that the certificate template is published on. Note that it may be necessary to double escape special characters in some scenarios.

IMPORTANT NOTE: When using HTTPS for the protocol, the Keyfactor SSL certificate or chain should be included in the trusted root store of the Hashicorp server to avoid TLS connection failures due to an untrusted certificate chain.

5. Set the environment for the plugin

Once the config file is created, it is necessary to set the appropriate environment variables so that the Keyfactor plugin is able to locate the config file. To do this, set a system variable for KF_CONF_PATH that points to the location of the config file. The method to accomplish this will vary based on your system configuration.

Using the EXPORT command, you can set the variable like this:

```
export KF_CONF_PATH='/usr/bin/keyfactor/config.json'
```

You may also be able to add this into your environment file or rc.local startup process based on your linux distribution.

If you are running Hashicorp Vault as a service with system, it may be necessary to add a line in the service config file so that this variable is set prior to Vault startup. This line would probably look something like this:

```
ExecStartPre=/bin/bash export KF_CONF_PATH='/usr/bin/keyfactor/config.json'
```

6. Install and register the plugin

To install and register the plugin on the Vault server, follow these steps:

1. Copy the Keyfactor plugin binary into a directory on the Vault server using SFTP or another file copy process. The location of this file does not really matter, but typically would be in a Vault plugins directory. An example

plugins path may look like this:
/usr/bin/keyfactor/vault-guides/secrets/engine/vault/plugins

2. Set the file to be executable.

```
chmod +x keyfactor
```

3. Get the sha256 checksum for the Keyfactor plugin binary file. This value will change with every version of the file. To get the sha256 checksum for the binary, run the following command:

```
sha256sum keyfactor
```

The result will show the sha256 value that will be needed in the next step.

```
47f549d44ab2abcb528aa45725b3a83334a9465bb487f3d1182add55e5580c36  keyfactor
```

4. Registering the new secrets engine within Vault is done using a vault write command to register the plugin into the Vault secrets catalog. (Note: if this command does not work, try removing the "/secret" part – vault version-dependent)

```
vault write sys/plugins/catalog/secret/keyfactor
sha256=47f549d44ab2abcb528aa45725b3a83334a9465bb487f3d1182add55e5580c36 command="keyfactor"
```

The result should look like this:

```
Success! Data written to: sys/plugins/catalog/secret/keyfactor
```

7. Enable the plugin within Hashicorp Vault

Once the plugin is installed into Vault, it just needs to be enabled. As a part of this enable process, you must specify the endpoint name that will be used for the secrets engine. This name is arbitrary. For this example, we are using keyfactor as the enpoint name, but it can be named to match existing endpoints that you are looking to replace with a connection to Keyfactor if necessary.

This example illustrates enabling a secrets engine on the endpoint called "keyfactor":

```
vault secrets enable keyfactor
```

a successfully enabled plugin will result in the following message:

```
Success! Enabled the keyfactor secrets engine at: keyfactor/
```

Using the Keyfactor plugin

1. Adding Roles

Hashicorp Vault supports being able to add roles to control certificate issuance policies such as allowed domains. To create a role, use the vault write command as noted below.

```
vault write keyfactor/roles/hashiwebserver allowed_domains=thedemodrive.com allow_subdomains=true
```

2. Issuing certificates

When requesting a certificate using the Keyfactor plugin, the command is the same as if you were issuing the certificate through the vault integrated PKI. As a part of the write command you will specify the role name you would like to use, as well as the common name on the certificate. A typical certificate issuance command is listed below for the hashiwebserver role, and a CN of foo.thedemodrive.com on the certificate.

Example:

```
vault write keyfactor/issue/hashiwebserver common_name=foo.thedemodrive.com
```

The resulting response will show the certificate data response for the request. This certificate will also be stored in the Vault secrets store.

```
Key                 Value
---                 -----
certificate         -----BEGIN CERTIFICATE-----
MIIFbzCCA1egAwIBAgITYwAAMFItaDXJJhODTwAAAAAwUjANBgkqhkiG9w0BAQ0FADA8MRYwFAYDVQQKEw1LZXlmYWN0b3IgSW5jMSIwI
AYDVQQDExlLZXlmYWN0b3IgVGVzdCBEcml2ZSBDQSAyMB4XDTIwMDgyNzExMDkwNFoXDTIwMDkwMzExMDkwNFowIDEeMBwGA1UEAxMVdG
VzdC50aGVkZW1vZHJpdmUuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAq4Rmh+IefQDSCMvZAisFlxKBXFcga+9YDI9
RVvKP3IketLCBRFbcfzx+qej/09xwYaXuKELU+y29kqHgfLFduUInYCbCiz+SZxjlEiXIJH5kqdIs86cHcY8BsuDIX+q4JYHaynAUm6+a
B1S0LhY/fEGDtszWSYVdRuXPGbQ9ZQVKrzNq2UwyS2/mZhSg/Cuwp/C0SCSd5ktz3wSFUl6oq8BKpOdkoWfYOBpaDZ3Cx/fItgiuir4C+
V4utHujXfXcfi7SPnk/d66EsH4hkQcn1esorTiiD5usLr9ZrUyMKBybbOYSDrutOVkcLJt8+6icjOArpnGkK/IsxkUTaAi25wIDAQABo4
IBhDCCAYAwHQYDVR0OBBYEFAKcqQqliw+Ll/gR9PzQXt3P37WmMB8GA1UdIwQYMBaAFMuGjbNJ6F1NO4GLe7ZmjvWjST0bMFgGA1UdHwR
RME8wTaBLoEmGR2h0dHA6Ly9rZXlmYWN0b3IudGhlZGVtb2RyaXZlLmNvbS9LZXlmYWN0b3IlMjBUZXN0JTIwRHJpdmUlMjBDQSUyMDIu
Y3JsMGMGCCsGAQUFBwEBBFcwVTBTBggrBgEFBQcwAoZHaHR0cDovL2tleWZhY3Rvci50aGVkZW1vZHJpdmUuY29tL0tleWZhY3RvciUyM
FRlc3QlMjBEcml2ZSUyMENBJTIwMi5jcnQwDgYDVR0PAQH/BAQDAgWgMD0GCSsGAQQBgjcVBwQwMC4GJisGAQQBgjcVCIb0k3GFpeF3gf
mRP4ah/HCE+fEhNoTBonqHk8wGAgFkAgEFMBMGA1UdJQQMMAoGCCsGAQUFBwMBMBsGCSsGAQQBgjcVCgQOMAwwCgYIKwYBBQUHAwEwDQY
JKoZIhvcNAQENBQADggIBAI19bqj9losB1LLSFOmNmlHpPsnpTpVUOUUA3XVUG8CQA1g2uu2rHdSJiExKYYjdYhMOPvMmGsU7QPoB3mxw
5I9eqfGEPD6WlIT0/wBX1wIBWa69+Y3cRFTx/wwOftWA9Vj7dQEt64GbhQZjMUR/RP6/02Bg0KJTMR46TRgFUs9QK1khKK+A2LTJuwKCx
vMzUlN2sPIKiE5B9lkAOd39cS8eGQ1GnCqXk8/5wFM8ohp1cmuO2BkKIg7Q43qujtll6qeM/tsU1zSZ4zKaPu4TuangeuKYZvniAIO0Lp
VVMivzNJMDu7Khyz3+CVZQrgr3yogFBK1fk7VO3jSntYQsg/ZOxQJtCJfv3gLhhEsvLNMxpMVjZ2R3z7brLNI52zptqizg16Gfbj4aA3d
ImkcwQfWCaaqagUsN9E9mgV6f1GE43c8LydMieJCxfsBF2Hmfl5zATYWzCs9/pX/HCnDJ8PvUnUyA9952CuRdwxyfFZ1ZWfEF6H6L8uVD
gwRxNIoQAGWtM45/wFv+qpbM882tV0/M/kgWlMuqPL+fwg412c87Q/cY3N0ibtU9IiWCBk7HNyYYDbj76E1HnfK8tl2lS8EVizflCS48q
Tt83EYdkevZaWaNbjPG2oCKe6nRqijKVLA3c4pFuJxwkyWTOeHDP3Y8RiFKuPVHlIj//mvCSVOg
-----END CERTIFICATE-----
issuing_ca          -----BEGIN CERTIFICATE-----
MIIIPTCCBiWgAwIBAgIQSFFVLWwLvrhJpr2mQUHwzDANBgkqhkiG9w0BAQ0FADA8MRYwFAYDVQQKEw1LZXlmYWN0b3IgSW5jMSIwIAYDV
QQDExlLZXlmYWN0b3IgVGVzdCBEcml2ZSBDQSAyMB4XDTE5MDIwNDIxMzE1N1oXDTQ0MDIwNDIxNDE1NlowPDEWMBQGA1UEChMNS2V5Zm
FjdG9yIEluYzEiMCAGA1UEAxMZS2V5ZmFjdG9yIFRlc3QgRHJpdmUgQ0EgMjCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAMU
CNOCm4TYBWKSVz6TWw+5diElexSHMi8YaVtR+g3D8VOeijoIKby59VNMhjBpTqYUE7HNm/X+cqvndlVIrih2stCEUuvJFtYDnG+qGrjdC
5mquGpKug1PQcrR/mC5ppNmJ2XhS5qzXurK3f9CtorHj0Noe0ZWHXUXwUWVQXKW22PNCRRp1eIZimRWUCvmMJbFoHKyhqHPt5rDAFRjM1
Sq94kl6zPqo+6HIP0eS8DyO93GAZMJRX+9TtvNTRmsLW/m4gYXFDTW+FhWfF+UZEE1hF24LbejFJb1uA7+UvdOvM0BkKuj/fbk3pcV33d
NhlKF5jvls3DYRIkboosyv0huyuLGG7/CDSba2BlJkCnKukhBdhhvTkWsM+rclIk7fQuUajo7nlqjRe7qxc/Ani3/0oQSEGgB7uaDXzoc
t37qUycIIkbgEEaJ20gU2h1dvHLFNfx7Jkfgq3ni0NQaKc4Vv4f3fwWhwEzkUZsf8OCkNhRvHuam1+ukiJmXNuf3iqp/bnSgznklwILiT
Fce4OmNd4bUoKtJSiEoSiTJ+KECeGBdQF0PtnKHSY61PvI7osHxmTK1pqIsemX6HvzYLARoSpTxHQ+qlD7AOqiN1RdlTHFoU/BNxlpF/9
Q0qUeQxERy/RAnh8bsZLICZEnBtKHKDIF1nwucnEM4aX2thvCTtAgMBAAGjggM5MIIDNTASBgNVHRMBAf8ECDAGAQH/AgEAMA4GA1UdDw
EB/wQEAwIBBjATBgkrBgEEAYI3FAIEBh4EAEMAQTAdBgNVHQ4EFgQUy4aNs0noXU07gYt7tmaO9aNJPRswEAYJKwYBBAGCNxUBBAMCAQA
wggLHBgNVHSAEggK+MIICujCCArYGCysGAQQBg507BQECMIICpTCCAmwGCCsGAQUFBwICMIICXh6CAloAVABoAGkAcwAgAEMAZQByAHQA
aQBmAGkAYwBhAHQAaQBvAG4AIABBAHUAdABoAG8AcgBpAHQAeQAgACgAQwBBACkAIABpAHMAIABhACAAcAByAGkAdgBhAHQAZQAgAHIAZ
QBzAG8AdQByAGMAZQAuACAAIABDAGUAcgB0AGkAZgBpAGMAYQB0AGUAcwAgAGkAcwBzAHUAZQBkACAAYgB5ACAAdABoAGkAcwAgAEMAQQ
AgAGEAcgBlACAAZgBvAHIAIABpAG4AdABlAHIAbgBhAGwAIAB1AHMAZQAgAG8AbgBsAHkALgAgACAAQQBuAHkAIABuAG8AbgAtAGEAdQB
0AGgAbwByAGkAegBlAGQAIABwAGEAcgB0AHkAIABzAGgAYQBsAGwAIABuAG8AdAAgAHIAZQBsAHkAIABvAG4AIAB0AGgAaQBzACAAQwBB
ACAAZgBvAHIAIABhAG4AeQAgAHIAZQBhAHMAbwBuAC4AIAAgAEYAbwByACAAbQBvAHIAZQAgAGkAbgBmAG8AcgBtAGEAdABpAG8AbgAsA
CAAcABsAGUAYQBzAGUAIAByAGUAZgBlAHIAIAB0AG8AIAB0AGgAZQAgAEMAZQByAHQAaQBmAGkAYwBhAHQAZQAgAFAAcgBhAGMAdABpAG
```

```
MAZQAgAFMAdABhAHQAZQBtAGUAbgB0ACAAYQB0ADoAIABoAHQAdABwADoALwAvAHAAbwBsAGkAYwB5AC4AawBlAHkAZgBhAGMAdABvAHI
AcABrAGkALgBjAG8AbQAvAGMAcABzAC4AaAB0AG0AbDAzBggrBgEFBQcCARYnaHR0cDovL3BvbGljeS5rZXlmYWN0b3Jwa2kuY29tL2Nw
cy5odG1sMA0GCSqGSIb3DQEBDQUAA4ICAQADytazW8H+sob+OcgYtPcuNMd8LxvVeX/gIlaLe7di9hUY+3aeNV9Itklp8z+y/SlLUZMVY
i2Fko0N3Yktstc7xqIeeYoZRJXVxGaLQAw2xrdjoBoCJV5DY9dQcpCEyt5cwxHGTXctOuLZLT38qlA+TBCZnhujcu/OsPYhKzNTPSNMoJ
A5MnRrWfFZYfbrou/m5/ysRYyrZeKX5kcDzSwZZvTRoJLi97YkZoo6PSwmt7JybmKUs+/PNPYMS3Dx45KurE822vGmfuIV7xt9URuTSaS
u8WRp9ghAjQ5jCTofXRsjgHmCiWPRzgZ36JNvJGsVLEAbahPN+27nBRFN9JIJQhI7+d0IpC44UnRyOuF7xSigXqrKXJ9t4cd2RJN916lW
Kim8TCpKBnpkjSAeLYUKtIivF2E+8I5fMrzYHtynZm7Hy1/MeqQMFBlUUEK+oj1rtr0+G0RKlJwRk3wHtiDKudHvsaNM9XsXTdlmliGDg
sAY17+eDdHzu81ZI/OYDtAa9k+i9BEkvkxhU/7474MEhl4codDCoiPcb0ew6raShZhsfwxNK/B/EOFQGhEAGivhJhSkUhiBFPIR8wrGkJ
DhlDNvbW1Jpkg9El6ixFzMT+mqkg1g5d/hkdz10cCMa89b3Z16AJYsLu2Q32nt5pZpTAY4jY+6LwriVr4u7qQjaQ==
-----END CERTIFICATE-----
private_key         -----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAq4Rmh+IefQDSCMvZAisFlxKBXFcga+9YDI9RVvKP3IketLCBRFbcfzx+qej/09xwYaXuKELU+y29kqHgfLFduUInY
CbCiz+SZxjlEiXIJH5kqdIs86cHcY8BsuDIX+q4JYHaynAUm6+aB1S0LhY/fEGDtszWSYVdRuXPGbQ9ZQVKrzNq2UwyS2/mZhSg/Cuwp/
C0SCSd5ktz3wSFUl6oq8BKpOdkoWfYOBpaDZ3Cx/fItgiuir4C+V4utHujXfXcfi7SPnk/d66EsH4hkQcn1esorTiiD5usLr9ZrUyMKBy
bbOYSDrutOVkcLJt8+6icjOArpnGkK/IsxkUTaAi25wIDAQABAoIBAHXLE2OFHu04sAbMgPglNcygL+mMCL83/F972h/9rGGIZmcvxUd6
5CoaEN9+HpyRCzl07NAHvh/XNRfMRtE8OqLt1P8K/5cEjPZzOXcyLXcqutWKe8bGUq1hyofgKpz9JYTU3r8jJHQbsIwSV0BDlUwv7laP7
SdHCV2UnJwJlhSW4rK31d9AP3l2X5M3Ax6y+Pmtdji743eEaaYixnplD9+BGlr0oTRg/TZ/UE828WNQF+/Mlnn20dR3u7IFKT39RQTiuw
l7nEYM8/pec5JMZvy1BvRjNlNv1eZ9ioCAysMWGEve+99JN5PCcCG//OE0OzONa0yqe7uec2HZmzD18dECgYEA2rrCFQ2eAKDhCdBnsKX
fKU0svi0wBFRF2wMR8fhYKQpiBFLkbrMmZmodHRi5wzZ2RrkFAu7BVqg3zKUfKWbb3wUVRzrs/1hXtdau2eS95j7838M1tcFHwNvveAO/
FZWVl2MZQdHaeLR3KeirjBkNsSAR2lohuECsSJaZAowoCZkCgYEAyL4vKRVuzjqbXmpmWKKSN4dpbhNvhb2XObBHwHoxNkiYDq60do2up
WDfVUBtnxR1O3AmBGFvhAMPQXcEAICg4XuUwGRmoJyCFgC0rb8uVxPIVRbXYRg1O12H64wg0HH6KvWyrIo6GBeBzpFnXPOMX62euSrV19
ROCUTGFNZMFH8CgYEAsWUGQ0zn+FqCKRN9BSeB9l1BDHxZlSlD/nxe8YAZADALjrYrzhw96JHnStHi1xA0nOcxyU8aPs3vc2n3+/wQFrB
osXx4+h8MA845wT5jRXmQXWpVBTPcne3CKfPf5gGLcVxN/7PTHFJA0xyBBP7Mu/rmf9DZyDWrhLOfJRHG5wECgYAi/Ts9HvY8TezMzSDf
rB5uPVT8Ebkrh3s3W+l1vBadzpNqY7siutlJDBSBSISS6L7ySD7oHo+QY/QhxfvVlpX0F1U7H4Tf08e8zAyyCBOsq88MPbn2u4Bzw36wa
LFDg8pF+KNW/ZTYpChE+AUbJ3w9JQb4YWX6g4/Cf9FCyoNiQwKBgFHcS5lmo8BJAcz8aKDPDJau0aqZU9MYQEQUOwMLZ+GouQeRP+mV2y
/sp2KV+bErKPPBEgV1gH1ly07k+zMjg7NbCSc1a1AMvfKeBlXemhhC8dl54cGMy5GLmAgefP4+qbuwfF+k9ANe5vPMuBztYwACPYv5azE
80+po4qrlVAm9
-----END RSA PRIVATE KEY-----
private_key_type    rsa
serial_number       63000030522D6835C92613834F000000003052
```

3. Reading certificates from the secrets store

After certificates are stored in the secrets store, you can then retrieve those certificates at a later time if necessary.

To list the certificates that exist within the Vault store, use the LIST option with vault. The only parameter that you need to include is the secrets store name for the store that you would like to read. The system will then return a list of all of the serial numbers for certificates that are present in that secrets store

```
vault list keyfactor
```

The results of the command will be a list of serial numbers for the certificates in that store location:

```
Keys
----
630000305345253543B8015BA5000000003053
63000030541DC9FF4893756253000000003054
6300003055AD38E7E2BD39B699000000003055
630000305602038A4CE30CFFF6000000003056
```

If you would like to retrieve a specific certificate from the store, you can do so by using the "vault read" command, and specifying the serial number of the certificate that you would like returned. The format for the command looks like this: vault read keyfactor/<serial>

Example:

```
vault read keyfactor/63000030522D6835C92613834F000000003052
```

The resulting response will show the certificate information for that certificate.

```
Key                                         Value
---                                         -----
63000030522D6835C92613834F000000003052
```

```
MIIFbzCCA1egAwIBAgITYwAAMFItaDXJJhODTwAAAAAwUjANBgkqhkiG9w0BAQ0FADA8MRYwFAYDVQQKEw1LZXlmYWN0b3IgSW5jMSIwI
AYDVQQDExlLZXlmYWN0b3IgVGVzdCBEcml2ZSBDQSAyMB4XDTIwMDgyNzExMDkwNFoXDTIwMDkwMzExMDkwNFowIDEeMBwGA1UEAxMVdG
VzdC50aGVkZW1vZHJpdmUuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAq4Rmh+IefQDSCMvZAisFlxKBXFcga+9YDI9
RVvKP3IketLCBRFbcfzx+qej/09xwYaXuKELU+y29kqHgfLFduUInYCbCiz+SZxjlEiXIJH5kqdIs86cHcY8BsuDIX+q4JYHaynAUm6+a
B1S0LhY/fEGDtszWSYVdRuXPGbQ9ZQVKrzNq2UwyS2/mZhSg/Cuwp/C0SCSd5ktz3wSFUl6oq8BKpOdkoWfYOBpaDZ3Cx/fItgiuir4C+
V4utHujXfXcfi7SPnk/d66EsH4hkQcn1esorTiiD5usLr9ZrUyMKBybbOYSDrutOVkcLJt8+6icjOArpnGkK/IsxkUTaAi25wIDAQABo4
IBhDCCAYAwHQYDVR0OBBYEFAKcqQqliw+Ll/gR9PzQXt3P37WmMB8GA1UdIwQYMBaAFMuGjbNJ6F1NO4GLe7ZmjvWjST0bMFgGA1UdHwR
RME8wTaBLoEmGR2h0dHA6Ly9rZXlmYWN0b3IudGhlZGVtb2RyaXZlLmNvbS9LZXlmYWN0b3IlMjBUZXN0JTIwRHJpdmUlMjBDQSUyMDIu
Y3JsMGMGCCsGAQUFBwEBBFcwVTBTBggrBgEFBQcwAoZHaHR0cDovL2tleWZhY3Rvci50aGVkZW1vZHJpdmUuY29tL0tleWZhY3RvciUyM
FRlc3QlMjBEcml2ZSUyMENBJTIwMi5jcnQwDgYDVR0PAQH/BAQDAgWgMD0GCSsGAQQBgjcVBwQwMC4GJisGAQQBgjcVCIb0k3GFpeF3gf
mRP4ah/HCE+fEhNoTBonqHk8wGAgFkAgEFMBMGA1UdJQQMMAoGCCsGAQUFBwMBMBsGCSsGAQQBgjcVCgQOMAwwCgYIKwYBBQUHAwEwDQY
JKoZIhvcNAQENBQADggIBAI19bqj9losB1LLSFOmNmlHpPsnpTpVUOUUA3XVUG8CQA1g2uu2rHdSJiExKYYjdYhMOPvMmGsU7QPoB3mxw
5I9eqfGEPD6WlIT0/wBX1wIBWa69+Y3cRFTx/wwOftWA9Vj7dQEt64GbhQZjMUR/RP6/02Bg0KJTMR46TRgFUs9QK1khKK+A2LTJuwKCx
vMzUlN2sPIKiE5B9lkAOd39cS8eGQ1GnCqXk8/5wFM8ohp1cmuO2BkKIg7Q43qujtll6qeM/tsU1zSZ4zKaPu4TuangeuKYZvniAIO0Lp
VVMivzNJMDu7Khyz3+CVZQrgr3yogFBK1fk7VO3jSntYQsg/ZOxQJtCJfv3gLhhEsvLNMxpMVjZ2R3z7brLNI52zptqizg16Gfbj4aA3d
ImkcwQfWCaaqagUsN9E9mgV6f1GE43c8LydMieJCxfsBF2Hmfl5zATYWzCs9/pX/HCnDJ8PvUnUyA9952CuRdwxyfFZ1ZWfEF6H6L8uVD
gwRxNIoQAGWtM45/wFv+qpbM882tV0/M/kgWlMuqPL+fwg412c87Q/cY3N0ibtU9IiWCBk7HNyYYDbj76E1HnfK8tl2lS8EVizflCS48q
Tt83EYdkevZaWaNbjPG2oCKe6nRqijKVLA3c4pFuJxwkyWTOeHDP3Y8RiFKuPVHlIj//mvCSVOg
```

4. Removing certificates from the secrets store

Removing certificates from the certificate store that are no longer needed is a very similar format to reading the certificates from the store. Using the vault command with the DELETE option, and then specifying the certificate serial number will remove it from the secrets store.

Note: Removing from the secrets store does not remove the certificate from the Keyfactor database. The certificate (and private key if enabled) can be re-downloaded and retrieved from Keyfactor if necessary.

Example:

```
vault delete keyfactor/63000030522D6835C92613834F000000003052
```

This command would result in a message that indicates that the certificate was removed:

```
Success! Data deleted (if it existed) at: keyfactor/63000030522D6835C92613834F000000003052
```

5. Disabling the Keyfactor Secrets Engine plugin

If the Keyfactor plugin is no longer needed within an environment, it can be removed using the DISABLE option for the secrets configuration in vault. The only parameter that is required is the secrets engine endpoint name.

```
vault secrets disable keyfactor
```

A successful disable will return the following message:

```
Success! Disabled the secrets engine (if it existed) at: keyfactor/
```

Reference for Plugin Commands

Create/update role

```
vault write keyfactor/roles/<rolename> allowed_domains=<domain1>,<domain2> allow_subdomains=true
```

List roles

```
vault list keyfactor/roles
```

Read role

```
vault read keyfactor/roles/<rolename>
```

Delete role

```
vault delete keyfactor/roles/<rolename>
```

Request certificate

```
vault write keyfactor/issue/<rolename> common_name=<CN>
```

List certificates

```
vault list keyfactor
```

Read certificate

```
vault read keyfactor/<Serial Number>
```

(Note: Certificate serial numbers are provided in the output for enrollment and list commands)

Revoke certificate

```
vault write keyfactor/revoke serial_number=<serial>
```

Sign CSR

```
vault write keyfactor/sign/<rolename> csr=<csr>
```

Read CA cert

```
vault read keyfactor/ca
```

Read CA chain

```
vault read keyfactor/ca_chain
```

Additional notes for this plugin:

TTL management is not handled through the secrets engine. Validity period is determined by certificate template. Expiration time is not reported in the enrollment output

Vault Agent Configuration

The vault agent is a Hashicorp software component that mediates communication with Vault and other systems. One of its features is the ability to auto-renew local certificates through the Vault PKI. This also works with our secrets engine.
Note: This is a different software component than the Keyfactor Vault Orchestrator, produced by Hashicorp
Setup and operation are as follows:

1. Enable both onboard and keyfactor pki engines with the same role name, allowed_domains, and allow_subdomains parameters.

```
vault secrets enable pki
vault write pki/root/generate/internal common_name=vault.jdk.cms ttl=8760h
vault write pki/roles/jdk allowed_domains=jdk.cms allow_subdomains=true max_ttl=8760h
```

2. Configure an AD template *that is only good for 1 hour (the msft minimum)*
Set up the keyfactor engine per other instructions, with this AD template.

3. On the vault server, set up a policy that allows access to engines mounted at the "pki" path (https://learn.hashicorp.com/tutorials/vault/policies):
Create a file on the vault server machine called "pki-admin-policy.hcl" with the following content

```
path "pki/*" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}
```

4. Enable RBAC in vault and create an app with permissions from the specified policy (https://learn.hashicorp.com/tutorials/vault/approle) – role is "jdk" in this case

```
vault auth enable approle
vault policy write pkiAdmin pki-admin-policy.hcl
vault write auth/approle/role/jdk secret_id_ttl=8760h token_policies=pkiAdmin
```

5. Retrieve tokens for the new role

```
jdk@ubuntu:~/keyfactor/vault-guides/secrets/engine$ vault read auth/approle/role/jdk/role-id
Key       Value
---       -----
role_id   ef7e32c2-c92f-c9c2-7e49-b40037bb5680
jdk@ubuntu:~/keyfactor/vault-guides/secrets/engine$ vault write -f auth/approle/role/jdk/secret-id
Key                   Value
---                   -----
secret_id             9b600253-da96-a6b3-9a03-949f0faed1b3
secret_id_accessor    2b9312f7-f629-e4b8-1423-55d6d93f3785
```

6. Set up a vault agent machine, separate from the vault server, for steps 7-10 and 12-13
VM with any Ubuntu version
Vault installed
Create folder "/vault-agent" and work from this directory through step 9 below
Create file "role_id_file" and paste in the role_id value from above
Create file "secret_id_seed_file" and paste in the secret_id from above

7. Make a "secrets template" for the agent to request certificates (https://www.vaultproject.io/docs/agent/template#certificates)
Create file "certificate.ctmpl" with the following content, updating the role name (jdk) and common_name as needed:

```
{{ with secret "pki/issue/jdk" "common_name=test.jdk.cms" "ttl=1m"}}
{{ .Data.certificate }}
{{ end }}
```

8. Create an agent config file called config.hcl as follows, updating the address under the vault stanza to point to your vault server. Note the secret_id_file_path filename is different from the filename in step 5.

```
pid_file = "./pidfile"

auto_auth {
        method "approle" {
                config = {
                        role_id_file_path = "/vault-agent/role_id_file"
                        secret_id_file_path = "/vault-agent/secret_id_file"
                }
        }

        sink "file" {
                config = {
                        path = "/vault-agent/token"
                }
        }
}

cache {
        use_auto_auth_token = true
}

vault {
        address = "http://192.168.0.31:8200"
}

listener "tcp" {
         address = "127.0.0.1:8100"
         tls_disable = true
}

template {
  source      = "/vault-agent/certificate.ctmpl"
  destination = "/vault-agent/certificate.pem"
}
```

9. Launch vault agent – it deletes the secret_id_file on launch, so we recreate each time from the seed file created in step 5.

```
cp secret_id_seed_file secret_id_file && vault agent -config=./config.hcl
```

You should see output like the following:

```
==> Vault agent started! Log data will stream in below:
==> Vault agent configuration:

                …

2020/12/05 18:28:38.563533 [INFO] (runner) starting
2020-12-05T10:28:38.564-0800 [INFO]  auth.handler: renewed auth token
```

```
2020/12/05 18:28:40.165700 [INFO] (runner) rendered "/vault-agent/certificate.ctmpl" => "/vault-agent/certificate.pem"
```

10. View the certificate created and observe that it's issued from the vault pki. If the agent is left running, it will auto-renew every few seconds, and re-running the below will show the updated "not after" times

```
openssl x509 -noout -text -in /vault-agent/certificate.pem
```

11. Back on the vault server, swap in the keyfactor engine for the onboard pki:

```
vault secrets move pki onboardPKI && vault secrets move keyfactor pki
```

12. Switch back to the agent machine, and observe that the next time it auto-renders the template, it gets a cert from the keyfactor API.
After the first keyfactor cert comes back, it takes about 5 minutes for the auto-renew to kick off a replacement for the new cert that expires in ~50 mins.
With the agent continuing to run, it should keep outputting the "rendered … => …" line, and openssl should keep showing new certs after each render.

13. Request certs from the agent machine CLI as follows, using whatever address is in the tcp listener from step 7 and whatever role and CN are needed. The write command should work against both onboard and keyfactor engines:

```
export VAULT_AGENT_ADDR="http://127.0.0.1:8100"
vault write pki/issue/jdk common_name=test.jdk.cms
```

14. On the vault machine, reset the secrets-engine path mappings so that the kf engine is mounted at "keyfactor" and the onboard PKI is mounted at "pki" once again

```
vault secrets move pki keyfactor && vault secrets move onboardPKI pki
```

Additional Resources

Webinar – Using the Keyfactor Secrets Engine for Hashicorp Vault (https://info.keyfactor.com/secrets-engine-hashicorp-vault)

Hashicorp Vault + Keyfactor PKI-as-a-Service (https://www.keyfactor.com/wp-content/uploads/Keyfactor-Secrets-Engine-HashiCorp-Vault-Datasheet.pdf)

Future Enhancements

1. Compatibility with enrollment flows with a request set to “pending” and approved only later.
2. Ability to register multiple instances of the plugin at different paths in Vault with different config/API app.
3. Storage of Keyfactor credentials within Vault.
4. Ability to configure a plugin without setting an environment variable and restarting Vault.
5. Source code publication and publicly-hosted binaries.