VMware Trust Authority and Entrust KeyControl

Integration Guide

27 May 2022

Contents

1. Introduction

This guide describes the integration of VMware Trust Authority with the Entrust KeyControl Key Management Solution (KMS). Entrust KeyControl can serve as a KMS in vCenter using the open standard Key Management Interoperability Protocol (KMIP).

The process starts by configuring vSphere Trust Authority services to attest your ESXi hosts, which then become capable of performing trusted cryptographic operations.

Also refer to the following document in the VMware online documentation:

- How vSphere Trust Authority protects your environment.

1.1. Documents to read first

This guide describes how to configure the Entrust KeyControl server as a KMS in vCenter.

To install and configure the Entrust KeyControl server as a KMIP server, see the `Entrust KeyControl nShield HSM Integration Guide`. You can access this in the Entrust Document Library.

Also refer to the following documents in the VMware online documentation:

- Using Encryption in a vSAN Cluster.
- Virtual Machine Encryption.

1.2. Requirements for vSphere Trust Authority

To use vSphere Trust Authority, your vSphere environment must meet these requirements:

- ESXi Trusted Host hardware requirements:
  - TPM 2.0
  - Secure boot must be enabled
  - EFI firmware
- Component requirements:
  - vCenter Server 7.0 or later
  - A dedicated vCenter Server system for the vSphere Trust Authority Cluster and ESXi hosts
  - A separate vCenter Server system for the Trusted Cluster and ESXi Trusted Hosts
  - Entrust KeyControl that has been deployed and configured. This will be the key server (called a Key Management Server, or KMS).

- Virtual machine requirements:
  - EFI firmware
  - Secure Boot Enabled

For more information see the VMWare Documentation on Prerequisites and Required Privileges for vSphere Trust Authority.

Entrust recommends that you allow only unprivileged connections unless you are performing administrative tasks.

1.3. Product configuration

Product	Version
KeyControl	5.5
vCenter Server	7.0.1 Build: 16858589
ESXi	ESXi-7.0U3c-19193900-standard

2. Procedures

2.1. Prerequisites

- Entrust KeyControl has been deployed and configured.
- VMware vSphere has been deployed and configured using vCenter.
- You have administrator rights to manage the KMS configuration in vCenter.

2.2. Configure Trust Authority

The procedures described here are a summary of the instructions found in the VMware documentation site. Refer to VMware documentation for more detailed instructions on Configuring vSphere Trust Authority.

When you configure vSphere Trust Authority, you must complete setup tasks on both the Trust Authority Cluster and the Trusted Cluster (Workload cluster). Some of these tasks are order-specific. Use the task sequence outlined in this guide.

2.3. Set up your workstation

To configure a vSphere Trust Authority deployment, you must first prepare a workstation with the necessary software and setup. Refer to VMware documentation for more details on Setting up Your Workstation.

The workflow in this guide uses a Windows 10 server. Most of the commands in this guide will be executed in the PowerShell, unless otherwise noted.

1. Install PowerCLI 12.1.0:

```
% Install-Module -Name VMware.PowerCLI -RequiredVersion 12.1.0.17009493 -AllowClobber
```

2. Verify that Microsoft .NET Framework 4.8 or later is installed.

   For information on installing .NET Framework 4.8, visit the Download .NET Framework 4.8 site.

3. Create a local folder in which to save the Trust Authority information that you export as files:

```
% mkdir c:\vta
% cd c:\vta
```

2.4. Enable the Trust Authority Administrator

To enable vSphere Trust Authority, you must add a user to the vSphere TrustedAdmins group. This user becomes the Trust Authority Administrator. You use the Trust Authority Administrator for most vSphere Trust Authority configuration tasks.

Check the VMware documentation for more detailed information on how to Enable the Trust Authority Administrator.

2.5. Enable the Trust Authority State

Making a vCenter Server cluster into a vSphere Trust Authority Cluster (also called enabling the Trust Authority State) starts the required Trust Authority services on the ESXi hosts in the cluster.

Check the VMware documentation for more detailed instructions on how to Enable the Trust Authority State.

1. Connect as the Trust Authority Administrator user to the vCenter Server of the Trust Authority Cluster:

   ```
   % Connect-VIServer -server TrustAuthorityCluster_VC_ip_address -User trust_admin_user -Password 'password'
   ```

   For example:

   ```
   % Connect-VIServer -server xx.xxx.xxx.11 -User administrator@vsphere.local -Password 'xxxxxxxx'

   Name                           Port  User
   ----                           ----  ----
   xx.xxx.xx.11                   443   VSPHERE.LOCAL\Administrator
   ```

   If you see the following error:

   ```
   Connect-VIServer : 2/14/2022 1:50:42 PM Connect-VIServer Error: Invalid server certificate. Use
   Set-PowerCLIConfiguration to set the value for the InvalidCertificateAction option to Prompt if you'd like to
   connect
   once or to add a permanent exception for this server.
   Additional Information: Could not establish trust relationship for the SSL/TLS secure channel with authority
   'xx.xxx.xxx.11'.
   ```

   Run the following command and try again:

   ```
   % Set-PowerCLIConfiguration -InvalidCertificateAction Ignore
   ```

2. Check the current state of the cluster.

```
% Get-TrustAuthorityCluster

Name                State               Id
----                -----               --
vSphere 7.0U1 Wit... Disabled            TrustAuthorityCluster-domain-c92
Tanzu VSAN Cluster  Disabled            TrustAuthorityCluster-domain-c5081
HA Cluster          Disabled            TrustAuthorityCluster-domain-c13030
VTA Cluster         Disabled            TrustAuthorityCluster-domain-c21111
```

The output shows either `Disabled` or `Enabled` in the State column for each cluster found. `Disabled` means that the Trust Authority services are not running.

3. Enable the Trust Authority Cluster.

```
% Set-TrustAuthorityCluster -TrustAuthorityCluster 'VTA Cluster' -State Enabled

Confirmation
Setting TrustAuthorityCluster 'VTA Cluster' with new State 'Enabled'. Do you want to proceed?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): A

Name                State               Id
----                -----               --
VTA Cluster         Enabled             TrustAuthorityCluster-domain-c21111
```

Two services start on the ESXi hosts in the Trust Authority Cluster:

- The Attestation Service.
- The Key Provider Service.

2.6. Collect information about ESXi hosts and vCenter server to be trusted

To establish trust, the vSphere Trust Authority Cluster requires information about the Trusted Cluster's ESXi hosts and vCenter Server. This information should be exported as files for importing into the Trust Authority Cluster. The information collected here are for the ESXi hosts in the Trusted Cluster vcenter Server (Workload).

Check the VMware documentation for detailed instructions on how to Collect Information about ESXi Hosts and vCenter Server to be Trusted.

2.6.1. Export the ESXi host description of software (the ESXi image)

1. Disconnect any current connection:

```
% Disconnect-VIServer -server * -Confirm:$false
```

2. Connect as the root user to one of the ESXi hosts in the Trusted Cluster (Workload Cluster):

```
% Connect-VIServer -server host_ip_address -User root -Password 'password'
```

For example:

```
% Connect-VIServer -server xx.xxx.xxx..201 -User root -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xxx..201                443   root
```

3. Confirm the ESXi host:

```
% Get-VMHost

Name                 ConnectionState PowerState NumCpu CpuUsageMhz CpuTotalMhz   MemoryUsageGB   MemoryTotalGB
Version
----                 --------------- ---------- ------ ----------- -----------   -------------   -------------
-------
xx.xxx.xxx..201      Connected       PoweredOn      16         360       33520          13.190          15.706
7.0.2
```

4. Assign `Get-VMHost` to a variable:

```
% $vmhost = Get-VMHost
```

5. Export the ESXi host description of software (the ESXi image).

   By default, this command exports the information to the `image.tgz` file. The `image.tgz` file in the example is named `image-esxi-201.tgz` to indicate what ESXi host this is from. Ensure that the destination directory exists before running this command.

```
% Export-VMHostImageDb -VMHost $vmhost -FilePath C:\vta\image-esxi-201.tgz

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/15/2022  11:53 AM          41046 image-esxi-201.tgz
```

The `Export-VMHostImageDb` cmdlet also works if you prefer to log in to the vCenter Server of the Trusted Cluster. Repeat for each ESXi version in the cluster that you want to trust. Use a different file name for each version so that you do not overwrite a previously exported file.

2.6.2. Export the ESXi host TPM CA Certificate

1. Assign `Get-Tpm2EndorsementKey -VMHost $vmhost` to a variable:

```
% $tpm2 = Get-Tpm2EndorsementKey -VMHost $vmhost
```

2. Export the CA certificate of a given TPM manufacturer.

   By default, this command exports the TPM certificate to the `cacert.zip` file. The `cacert.zip` file in the example is named `tpmcacert-201.zip` to indicate what ESXi host this is from. Ensure that the destination directory exists before running this command.

```
% Export-Tpm2CACertificate -Tpm2EndorsementKey $tpm2 -FilePath C:\vta\tpmcacert-201.zip

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/15/2022  11:51 AM           2858 tpmcacert-201.zip
```

3. Repeat for each TPM hardware type in the cluster that you want to trust.

   Use a different file name for each TMP hardware type so that you do not overwrite a previously exported file.

If you have issues exporting the TPM2 CA Certificate of the EXSi host, with errors like this:

```
Export-Tpm2CACertificate  Invalid X509Certificate2 provided, its CA issuer file could not be downloaded.
```

You can Collect the TPM Endorsement Key instead. ---

2.6.3. Collect the TPM Endorsement Key

This method should only be used if you failed to Export the ESXi host TPM CA Certificate.

Check the VMware documentation for detailed instructions on how to Collect the TPM Endorsement Key.

You can export a TPM endorsement key (EK) certificate from an ESXi host, and import it to the vSphere Trust Authority Cluster. You do so when you want to trust an individual ESXi host in the Trusted Cluster.

To import a TPM EK certificate into the Trust Authority Cluster, you must change the Trust Authority Cluster's default attestation type to accept EK certificates. The default attestation type accepts TPM Certificate Authority (CA) certificates. Some TPMs do not include EK certificates. If you want to trust individual ESXi hosts, the TPM must include an EK certificate.

2.6.3.1. Change the Trust Authority Cluster's default attestation type to accept EK certificates

1. Disconnect from the ESXi host:

```
% Disconnect-VIServer -server * -Confirm:$false
```

2. Connect as the Trust Authority Administrator user to the vCenter Server of the Trust Authority Cluster.

```
% Connect-VIServer -server TrustAuthorityCluster_VC_ip_address -User trust_admin_user -Password 'password'
```

   For example:

```
% Connect-VIServer -server xx.xxx.xxx.11 -User administrator@vsphere.local -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xx.11                   443   VSPHERE.LOCAL\Administrator
```

3. Check the current state of the cluster:

```
% Get-TrustAuthorityCluster

Name                 State                Id
----                 -----                --
vSphere 7.0U1 Wit... Disabled             TrustAuthorityCluster-domain-c92
Tanzu VSAN Cluster   Disabled             TrustAuthorityCluster-domain-c5081
HA Cluster           Disabled             TrustAuthorityCluster-domain-c13030
VTA Cluster          Enabled              TrustAuthorityCluster-domain-c21111
```

4. Assign the `Get-TrustAuthorityCluster` cluster information to a variable:

```
% $VTA = Get-TrustAuthorityCluster 'VTA Cluster'
```

5. Assign the `Get-TrustAuthorityTpm2AttestationSettings` information to a variable:

```
% $tpm2Settings = Get-TrustAuthorityTpm2AttestationSettings -TrustAuthorityCluster $VTA
```

6. Run the `Set-TrustAuthorityTpm2AttestationSettings` cmdlet, specifying `RequireEndorsementKey`. At the confirmation prompt, press `Enter`. (The default is `Y`.)

```
% Set-TrustAuthorityTpm2AttestationSettings -Tpm2AttestationSettings $tpm2Settings -RequireEndorsementKey

Confirmation
Configure the Tpm2AttestationSettings 'TrustAuthorityTpm2AttestationSettings-domain-c2111' with the following
parameters:
 RequireCertificateValidation: False
 RequireEndorsementKey: True
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

Name                                 RequireEndorsementKey         RequireCertificateValidation   Health
----                                 ---------------------         ----------------------------   ------
TrustAuthorityTpm2AttestationSettings... True                      False                          Ok
```

2.6.3.2. Export the TPM EK certificate

You need to do this for every ESXi host you want trust in the Trusted cluster.

1. Disconnect any current connection:

```
% Disconnect-VIServer -server * -Confirm:$false
```

2. Connect as the root user to one of the ESXi hosts in the Trusted Cluster (Workload Cluster):

```
% Connect-VIServer -server host_ip_address -User root -Password 'password'
```

   For example:

```
% Connect-VIServer -server xx.xxx.xxx..201 -User root -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xxx..201                443   root
```

3. Confirm the ESXi host:

```
% Get-VMHost

Name                 ConnectionState PowerState NumCpu CpuUsageMhz CpuTotalMhz   MemoryUsageGB   MemoryTotalGB
Version
----                 --------------- ---------- ------ ----------- -----------   -------------   -------------
-------
xx.xxx.xxx..201      Connected       PoweredOn      16         360       33520          13.190          15.706
7.0.2
```

4. Assign `Get-VMHost` to a variable:

```
% $vmhost = Get-VMHost
```

5. Export the EK certificate of the ESXi host:

```
% Export-Tpm2EndorsementKey -VMHost $vmhost -FilePath -FilePath C:\vta\tpm2ek-201.json
```

The file is created. Repeat the process for the other ESXi hosts you want to trust. Make sure the filename is different for each ESXi host.

2.6.4. Collect information about the Trusted Cluster's vCenter server

Export the Trusted Cluster's vCenter Server principal information.

1. Disconnect from the ESXi host:

   ```
   % Disconnect-VIServer -server * -Confirm:$false
   ```

2. Connect to the vCenter Server of the Trusted Cluster using the Trust Authority Administrator user.

   ```
   % Connect-VIServer -server TrustedCluster_VC_ip_address -User trust_admin_user -Password 'password'
   ```

   For example:

   ```
   % Connect-VIServer -server 10.194.148.12 -User administrator@hpz8.local -Password 'xxxxxxx'

   Name                           Port  User
   ----                           ----  ----
   xx.xxx.xxx..12                 443   HPZ8.LOCAL\Administrator
   ```

3. Export the Trusted Cluster's vCenter Server principal information:

   ```
   % Export-TrustedPrincipal -FilePath C:\vta\trustedcluserprincipal-12.json

   Mode                 LastWriteTime         Length Name
   ----                 -------------         ------ ----
   -a----         2/15/2022  11:57 AM           1867 trustedcluserprincipal-12.json
   ```

4. (Optional) If you want to trust an individual host, you must export the TPM EK public key certificate.

   See Export and Import a TPM Endorsement Key Certificate on VMware documentation site for more details.

   The following files are created:

   - TPM CA certificate file (`.zip` file extension)
   - ESXi image file (`.tgz` file extension)
   - vCenter Server principal file (`.json` file extension)

2.7. Import the Trusted Host Information to the Trust Authority Cluster

You import the exported ESXi host and vCenter Server information into the vSphere Trust Authority Cluster, so that the Trust Authority Cluster knows which hosts it can attest.

Check the VMware documentation for more details on how to Import the Trusted Host Information to the Trust Authority Cluster.

1. Disconnect from all connections:

```
%  Disconnect-VIServer -server * -Confirm:$false
```

2. Ensure that you are connected as the Trust Authority Administrator to the vCenter Server of the Trust Authority Cluster:

```
% Connect-VIServer -server TrustAuthorityCluster_VC_ip_address -User trust_admin_user -Password 'password'
```

   For example:

```
% Connect-VIServer -server xx.xxx.xxx..11 -User administrator@vsphere.local -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xxx.11                  443   VSPHERE.LOCAL\Administrator
```

3. Show the clusters managed by this vCenter Server:

```
% Get-TrustAuthorityCluster

Name                 State               Id
----                 -----               --
vSphere 7.0U1 Wit... Disabled            TrustAuthorityCluster-domain-c92
Tanzu VSAN Cluster   Disabled            TrustAuthorityCluster-domain-c5081
HA Cluster           Disabled            TrustAuthorityCluster-domain-c13030
VTA Cluster          Enabled             TrustAuthorityCluster-domain-c21111
```

4. Assign the `Get-TrustAuthorityCluster` cluster information to a variable:

```
% $VTA = Get-TrustAuthorityCluster 'VTA Cluster'
```

5. Import the vCenter Server principal information of the Trusted Cluster into the Trust Authority Cluster:

   For example, import the `principal.json` file previously exported in Collect information about ESXi hosts and vCenter server to be trusted:

```
% New-TrustAuthorityPrincipal -TrustAuthorityCluster $vTA -FilePath C:\vta\trustedcluserprincipal-12.json

Name                                    Domain          Type       TrustAuthorityClusterId
----                                    ------          ----       -----------------------
vpxd-411998e7-05f9-4d34-9c58-f258b12a05bb     hpz8.local      STS_USER   TrustAuthorityCluster-domain-c21111
```

6. Verify the import:

```
% Get-TrustAuthorityPrincipal -TrustAuthorityCluster $VTA

Name                                    Domain          Type       TrustAuthorityClusterId
----                                    ------          ----       -----------------------
vpxd-411998e7-05f9-4d34-9c58-f258b12a05bb     hpz8.local      STS_USER   TrustAuthorityCluster-domain-c21111
```

7. Import the Trusted Platform Module (TPM) CA certificate information:

   For example, import the TPM CA certificate information from the `cacert.zip` file previously exported in Collect information about ESXi hosts and vCenter server to be trusted:

   +

```
% New-TrustAuthorityTpm2CACertificate -TrustAuthorityCluster $VTA -FilePath C:\vta\tpmcacert-201.zip

TrustAuthorityClusterId                 Name                                    Health
-----------------------                 ----                                    ------
TrustAuthorityCluster-domain-c21111     CE77153B6E110CA4AE2971A09851EF499326202A Ok
```

If you exported the TPM endorsement key of the ESXi hosts instead of the TPM CA Certificate and you changed the Trust Authority Cluster's default attestation type to accept EK certificates, import the TPM endorsement key of each ESXi host instead. For example:

```
% New-TrustAuthorityTpm2EndorsementKey -TrustAuthorityCluster $VTA -FilePath C:\vta\tpm2ek-201.json
```

Repeat the process for each ESXi host.

8. Import the ESXi host base image information:

   For example, import the image information from the `image.tgz` file previously exported in Collect information about ESXi hosts and vCenter server to be trusted:

```
% New-TrustAuthorityVMHostBaseImage -TrustAuthorityCluster $VTA -FilePath C:\vta\image-esxi-201.tgz

TrustAuthorityClusterId                 VMHostVersion                           Health
-----------------------                 -------------                           ------
TrustAuthorityCluster-domain-c21111     ESXi 7.0.2-0.0.17867351                 Ok
```

The Trust Authority Cluster knows which ESXi hosts it can remotely attest, and so,

which hosts it can trust.

2.8. Create the Key Provider on the Trust Authority Cluster

For the Key Provider Service to connect to a key provider, you must create a trusted key provider then configure a trust setup between the vSphere Trust Authority Cluster and Entrust Key Control, the key server (KMS). This configuration involves setting up client and server certificates.

Check the VMware documentation for more detail information on how to Create the Key Provider on the Trust Authority Cluster.

2.8.1. Create and activate a primary key (master key) on the key server for the trusted key provider

This key wraps other keys and secrets used by this trusted key provider. The assumption here is that you have deployed an Entrust KeyControl cluster, as described in the *Entrust KeyControl nShield Integration Guide*.

1. Log into the KeyControl web interface using the tenant login URL.
2. Create a client certificate as described in the *Entrust KeyControl nShield Integration Guide* and download the certificate bundle.
3. Unzip the certificate bundle. You should have two files:
   a. `cacert.pem`.
   b. The SSL certificate `.pem` file named after the name given to the client certificate.
4. To create a key, use the `pykmip.py` python script referenced in this guide.
5. Install Python on your PowerShell:

```
% [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
% Invoke-WebRequest -Uri "https://www.python.org/ftp/python/3.7.0/python-3.7.0.exe" -OutFile "c:/temp/python-
3.7.0.exe"
% c:/temp/python-3.7.0.exe /quiet InstallAllUsers=0 PrependPath=1 Include_test=0
```

6. Once python is installed, install `pip` and then the python `pykmip` package:

```
% py -3 -m ensurepip
% py -3 -m pip install --upgrade pip
% py -3 -m pip install pykmip
```

7. Modify the `pykmip.py` file so it adheres to your environment. (Change KMIP server name and so on).
8. Copy the client certificate `.pem` files from KeyControl to the folder where your

`pykmip.py` file is located.

9. Rename the SSL certificate `.pem` file to `vta.pem`.

   You should have the following files in the folder:

   ```
   cacert.pem
   pykmip.py
   vta.pem
   ```

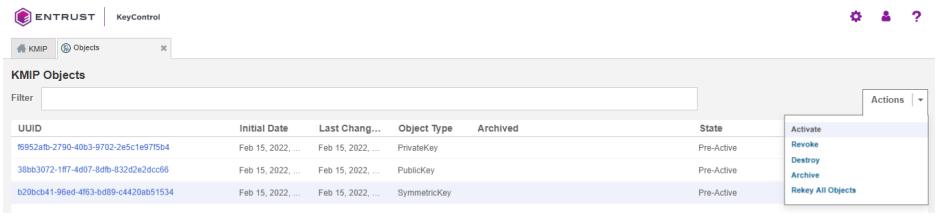
10. Run the python script to create a key:

    ```
    %  py -3 .\pykmip.py create_symkey

    C:\vta\pykmip\vta.pem C:\vta\pykmip/cacert.pem
    UUID: b20bcb41-96ed-4f63-bd89-c4420ab51534
    ```

    This should create a key in KeyControl with the UUID referenced above.

11. Log into the **KeyControl Tenant Administration** page, select the key and then select **Actions > Activate** to activate it:



The key is created and activated in KeyControl.

2.8.2. Create the key provider

1. Disconnect from all connections:

   ```
   %  Disconnect-VIServer -server * -Confirm:$false
   ```

2. Ensure that you are connected to the vCenter Server of the Trust Authority Cluster:

   ```
   % Connect-VIServer -server TrustAuthorityCluster_VC_ip_address -User trust_admin_user -Password 'password'
   ```

   For example:

   ```
   % Connect-VIServer -server xx.xxx.xxx..11 -User administrator@vsphere.local -Password 'xxxxxxxx'

   Name                           Port  User
   ----                           ----  ----
   xx.xxx.xxx.11                  443   VSPHERE.LOCAL\Administrator
   ```

3. Assign `Get-TrustAuthorityCluster` information to a variable called `$VTA`:

```
% $VTA = Get-TrustAuthorityCluster 'VTA Cluster'
```

4. Create the trusted key provider using the `New-TrustAuthorityKeyProvider` cmdlet. For this command:

   - Use the UUID of the keys created earlier and assign a name. For example, KeyControl1.
   - Use the IP address of the KeyControl server.
   - The PrimaryKeyID is normally a key ID that comes from the key server in the form of a UUID. Do not use the key name for PrimaryKeyID.
   - The `New-TrustAuthorityKeyProvider` cmdlet can take other options, such as `KmipServerPort`, `ProxyAddress`, and `ProxyPort`.
   - Each logical key provider must have a unique name across all vCenter Server systems.
   - To add multiple key servers to the key provider, use the `Add-TrustAuthorityKeyProviderServer` cmdlet.

   For example:

```
% New-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA -PrimaryKeyId b20bcb41-96ed-4f63-bd89-c4420ab51534 -Name
KeyControl1 -KmipServerAddress xx.xx.xx..160

Name                PrimaryKeyId         Type       TrustAuthorityClusterId
----                ------------         ----       -----------------------
KeyControl1         b20bcb41-96ed-4f6... KMIP       TrustAuthorityCluster-domain-c21111
```

5. Establish the trusted connection so that the KeyControl server trusts the trusted key provider:

   a. Obtain the trusted key providers in the given Trust Authority Cluster:

```
% $kp = Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA
```

   If you have more than one trusted key provider, use commands similar to the following to select the one you want:

```
Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA
# <The trusted key providers listing is displayed.>

$kp = Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA | Select-Object -Last 1
# Using Select-Object -Last 1 selects the last trusted key provider in the list.
```

   b. Upload the certificate and private key using the `Set-TrustAuthorityKeyProviderClientCertificate` command:

```
Set-TrustAuthorityKeyProviderClientCertificate -KeyProvider $kp -CertificateFilePath <path/to/certfile.pem>
-PrivateKeyFilePath <path/to/privatekey.pem>
```

Use the `privatekey.pem` (`vta.pem`) for both the `certfile.pem` and `privatekey.pem` files.

For example:

```
% Set-TrustAuthorityKeyProviderClientCertificate -KeyProvider $kp -CertificateFilePath C:\vta\vta.pem
-PrivateKeyFilePath C:\vta\vta.pem

Thumbprint                               Subject
----------                               -------
A031DE6E8BE1034771044B1DA96657CACD5D5DF2  CN=vta, O=HyTrust Inc., C=US
```

As a result, the trusted key provider has established trust with the key server.

6. Finish the trust setup by uploading a key server certificate so that the trusted key provider trusts the key server:
   a. Obtain the trusted key providers in the given Trust Authority Cluster:

```
% $kp = Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA
```

If you have more than one trusted key provider, use commands similar to the following to select the one you want:

```
Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA

# <The trusted key providers listing is displayed.>

$kp = Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA | Select-Object -Last 1

# Using Select-Object -Last 1 selects the last trusted key provider in the list.
```

   b. Get the key server server certificate:

```
% Get-TrustAuthorityKeyProviderServerCertificate -KeyProviderServer $kp.KeyProviderServers

Certificate                              Trusted    KeyProviderServerId       KeyProviderId
-----------                              -------    -------------------       -------------
[Subject]...                             False      domain-c21111-KeyContr... domain-c21111-KeyControl1
```

Initially, the certificate is not trusted, so the Trusted state is `False`. If you have more than one key server configured, a list of certificates is returned. Verify and add each certificate using the following instructions.

   c. Before trusting the certificate, assign `Get-TrustAuthorityKeyProviderServerCertificate -KeyProviderServer $kp.KeyProviderServers` information to a variable. Run the

`$cert.Certificate.ToString()` command and verify the output:

```
% $cert = Get-TrustAuthorityKeyProviderServerCertificate -KeyProviderServer $kp.KeyProviderServers
% $cert.Certificate.ToString()

[Subject]
  CN=keycontrol55-1f.interop.com, O=HyTrust Inc., C=US

[Issuer]
  CN=HyTrust KeyControl Certificate Authority, O=HyTrust Inc., C=US

[Serial Number]
  00A0AA8922

[Not Before]
  5/31/2011 8:00:00 PM

[Not After]
  12/31/2049 6:59:59 PM

[Thumbprint]
  1D591D20E8A03BC00DD548CE27498721E86F3B98
```

d. Add the KMIP server certificate to the trusted key provider:

```
% Add-TrustAuthorityKeyProviderServerCertificate -ServerCertificate $cert

Certificate                          Trusted    KeyProviderServerId       KeyProviderId
-----------                          -------    -------------------       -------------
[Subject]...                         True                                 domain-c21111-KeyControl1
```

7. Verify the status of the key provider:

a. Refresh the key provider status:

```
% $kp = Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA
```

If you have more than one trusted key provider, use commands similar to the following to select the one you want:

```
Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA

# <The trusted key providers listing is displayed.>

$kp = Get-TrustAuthorityKeyProvider -TrustAuthorityCluster $VTA | Select-Object -Last 1

# Using Select-Object -Last 1 selects the last trusted key provider in the list.
```

b. Get the key provider status:

```
% $kp.Status

KeyProviderId             Health HealthDetails ServerStatus
-------------             ------ ------------- ------------
domain-c21111-KeyControl1     Ok {}            {xx.xxx.xxx..160}
```

The status can take a few minutes to be refreshed. To view the status, reassign the `$kp` variable and rerun the `$kp.Status` command. A Health status of `Ok` indicates that the key provider is running correctly.

2.9. Export the Trust Authority Cluster information

For the Trusted Cluster to connect to the vSphere Trust Authority Cluster, you must export the Trust Authority Cluster's service information in the form of a file then import that file to the Trusted Cluster.

Check the VMware documentation for more details on how to Export the Trust Authority Cluster Information.

To export the Trust Authority Cluster information:

1. Disconnect from all connections:

```
%  Disconnect-VIServer -server * -Confirm:$false
```

2. Ensure that you are connected to the vCenter Server of the Trust Authority Cluster:

```
% Connect-VIServer -server TrustAuthorityCluster_VC_ip_address -User trust_admin_user -Password 'password'
```

   For example:

```
% Connect-VIServer -server xx.xxx.xxx..11 -User administrator@vsphere.local -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xxx.11                  443   VSPHERE.LOCAL\Administrator
```

3. Export the Trust Authority Cluster's Attestation Service and Key Provider Service information:

   This command exports the service information to the `clsettings.json` file:

```
% $VTA = Get-TrustAuthorityCluster 'VTA Cluster'

% Export-TrustAuthorityServicesInfo -TrustAuthorityCluster $VTA -FilePath C:\vta\clsettings.json

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/16/2022   9:45 AM           4183 clsettings.json
```

The file containing the Trust Authority Cluster information is created.

2.10. Import the Trust Authority Cluster information to the Trusted Hosts

After you have imported the vSphere Trust Authority Cluster information to the Trusted Cluster, the Trusted Hosts start the attestation process with the Trust Authority Cluster.

Check the VMware documentation for more details on how to Import the Trust Authority Cluster Information to the Trusted Hosts

To import the Trust Authority Cluster information:

1. Disconnect from all connections:

```
%  Disconnect-VIServer -server * -Confirm:$false
```

2. Ensure that you are connected as the Trust Authority Administrator to the vCenter Server of the Trusted Cluster (Workload cluster):

```
% Connect-VIServer -server TrustedCluster_VC_ip_address -User trust_admin_user -Password 'password'
```

   For example:

```
% Connect-VIServer -server xx.xxx.xxx..12 -User administrator@hpz8.local -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xxx.12                  443   HPZ8.LOCAL\Administrator
```

3. Verify that the state of the Trusted Cluster is `Disabled`:

```
% Get-TrustedCluster

Name                State               Id
----                -----               --
Workload Cluster    Disabled            TrustedCluster-domain-c1001
```

4. Assign the `Get-TrustedCluster` information to a variable and verify its value:

```
% $TC = Get-TrustedCluster -Name 'Workload Cluster'

% $TC

Name                State               Id
----                -----               --
Workload Cluster    Disabled            TrustedCluster-domain-c1001
```

5. Import the Trust Authority Cluster information to the vCenter Server:

   Use the `clsettings.json` file previously exported in Export the Trust Authority Cluster Information:

```
% Import-TrustAuthorityServicesInfo -FilePath C:\vta\clsettings.json

Confirmation
Importing the TrustAuthorityServicesInfo into Server 'xx.xxx.xxx.12'. Do you want to proceed?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

ServiceAddress                ServicePort         ServiceGroup
--------------                -----------         ------------
xx.xxx.xxx.199                443                 host-21115:4e7ddb95-198b-4f...
xx.xxx.xxx.199                443                 host-21115:4e7ddb95-198b-4f...
```

6. Enable the Trusted Cluster:

```
% Set-TrustedCluster -TrustedCluster $TC -State Enabled

Confirmation
Setting TrustedCluster 'Workload Cluster' with new TrustedState 'Enabled'. Do you want to proceed?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):

Name                State               Id
----                -----               --
Workload Cluster    Enabled             TrustedCluster-domain-c1001
```

   You can also enable the Trusted Cluster by enabling the Attestation Service and the Key Provider Service individually. Use the `Add-TrustedClusterAttestationServiceInfo` and `Add-TrustedClusterKeyProviderServiceInfo` commands. For example, the following commands enable the services one at a time for the cluster Trusted Cluster that has two Key Provider Services and two Attestation Services:

```
% Add-TrustedClusterAttestationServiceInfo -TrustedCluster 'Workload Cluster' -AttestationServiceInfo (Get-
AttestationServiceInfo | Select-Object -index 0,1)

% Add-TrustedClusterKeyProviderServiceInfo  -TrustedCluster 'Workload Cluster' -KeyProviderServiceInfo (Get-
KeyProviderServiceInfo | Select-Object -index 0,1)
```

7. Verify that the Attestation Service and the Key Provider Service are configured in the Trusted Cluster:

```
% $TC = Get-TrustedCluster -Name 'Workload Cluster'

% $TC.AttestationServiceInfo

ServiceAddress                ServicePort         ServiceGroup
--------------                -----------         ------------
xx.xxx.xxx.199                443                 host-21115:4e7ddb95-198b-4f...

% $TC.KeyProviderServiceInfo

ServiceAddress                ServicePort         ServiceGroup
--------------                -----------         ------------
xx.xxx.xxx.199                443                 host-21115:4e7ddb95-198b-4f...
```

The ESXi Trusted Hosts in the Trusted Cluster begin the attestation process with the Trust Authority Cluster.

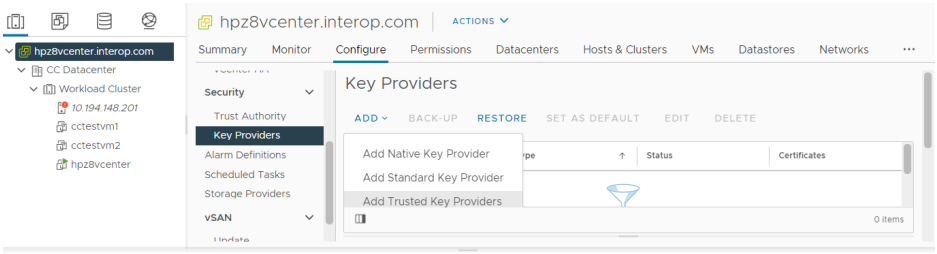
2.11. Configure the Trusted Key Provider for Trusted Hosts

You can configure the trusted key provider by using the vSphere Client or the Command Line.

2.11.1. Using the vSphere Client

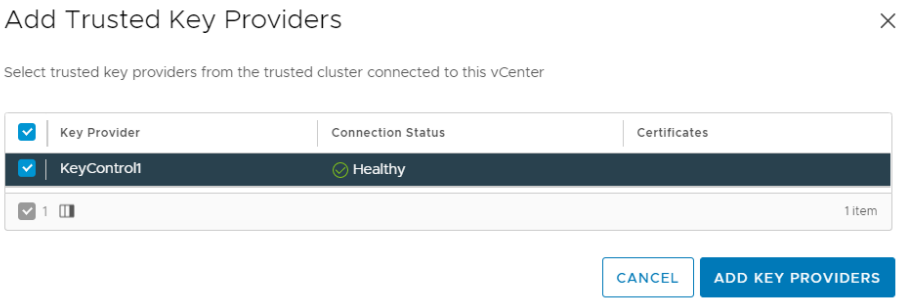
Check the VMware documentation for more details on how to Configure the Trusted Key Provider for Trusted Hosts Using the vSphere Client.

1. Connect to vCenter Server of the Trusted Cluster by using the vSphere Client.
2. Log in as the vCenter Server Administrator, or an Administrator that has the `Cryptographic operations.Manage key servers` privilege.
3. Select the **vCenter Server**, then select **Configure**.
4. Select **Key Providers** under **Security**.



5. Select **Add Trusted Key Providers**.

   The trusted key providers that are available are shown with a status of `Connected/Healthy`.



6. Select a trusted key provider and select **Add Key Providers**.
7. The trusted key provider is shown as Trusted and Connected. If this is the first trusted key provider that you add, it is marked as the default.

ESXi Trusted Hosts can now perform cryptographic operations, such as creating encrypted virtual machines.

2.11.2. Using the Command Line

Check the VMware documentation for more details on how to Configure the Trusted Key Provider for Trusted Hosts Using the Command Line.

1. Disconnect from all connections:

```
%  Disconnect-VIServer -server * -Confirm:$false
```

2. Ensure that you are connected as the Trust Authority Administrator to the vCenter Server of the Trusted Cluster (Workload cluster):

```
% Connect-VIServer -server TrustedCluster_VC_ip_address -User trust_admin_user -Password 'password'
```

   For example:

```
% Connect-VIServer -server xx.xxx.xxx..12 -User administrator@hpz8.local -Password 'xxxxxxxx'

Name                           Port  User
----                           ----  ----
xx.xxx.xxx.12                  443   HPZ8.LOCAL\Administrator
```

3. Obtain the trusted key provider:

```
% Get-KeyProvider
```

   You can use the `-Name` option to specify a single trusted key provider.

4. Assign the `Get-KeyProvider` trusted key provider information to a variable:

```
% $workload_kp = Get-KeyProvider
```

   If you have multiple trusted key providers, you can use Select-Object to select one of them:

```
% $workload_kp = Get-KeyProvider | Select-Object -Index 0
```

5. Register the trusted key provider:

```
% Register-KeyProvider -KeyProvider $workload_kp

Name                     DefaultForSystem     ClientCertificateExpiryDate
----                     ----------------     ---------------------------
KeyControl1              False                1/1/0001 12:00:00 AM
```

6. Register additional trusted key providers by repeating the previous steps.

7. Configure a default key provider:
    a. Set the default trusted key provider:

```
% Set-KeyProvider -KeyProvider $workload_kp -DefaultForSystem

Name                      DefaultForSystem     ClientCertificateExpiryDate
----                      ----------------     ---------------------------
KeyControl1               True                 1/1/0001 12:00:00 AM
```

    b. To set the key provider at the cluster level, run the following command:

```
% Add-EntityDefaultKeyProvider -KeyProvider $workload_kp -Entity 'Workload Cluster'
```

    c. To set the key provider at the folder level, run the following command:

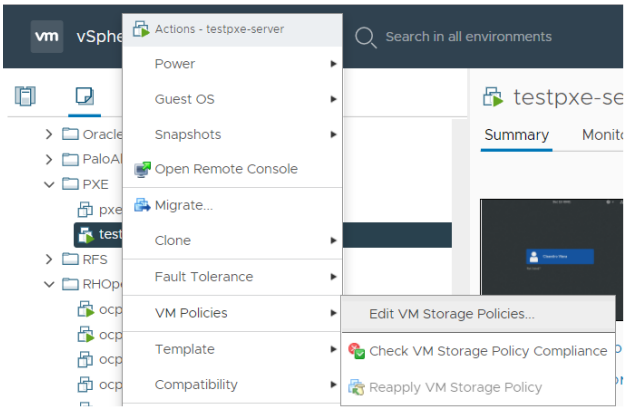
```
% Add-EntityDefaultKeyProvider -KeyProvider $workload_kp -Entity 'TC Folder'
```

3. Testing

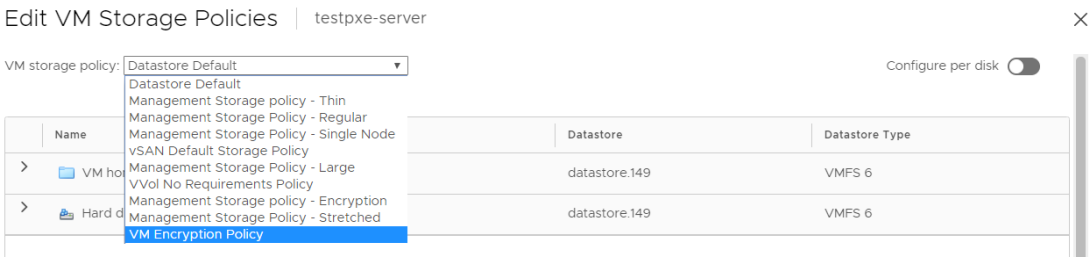
Encrypting a virtual machine with a trusted key provider looks the same as the virtual machine encryption user experience that was first delivered in vSphere. See Use Encryption in Your vSphere Environment.

This chapter demonstrates the encryption of a VM using VMware Trust Authority.

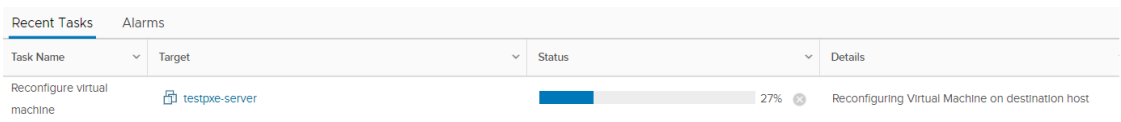
1. Connect to vCenter Server of the Trusted Cluster (Workload cluster) by using the vSphere Client.
2. Log in as the vCenter Server Administrator, or an Administrator that has the `Cryptographic operations.Manage key servers` privilege.
3. Locate a VM that you would like to encrypt.
4. Make sure the **Power** state for the VM indicates that it is powered off.
5. Right-click the VM for which you would like to enable encryption, and select **VMPolicies > Edit VM Storage Policies**.



6. Select the storage policy **VM Encryption Policy** and select **OK**.



This will trigger a reconfiguration of the VM.



Once the reconfiguration is complete, the disks are encrypted and the keys are managed by the configured KMS, in our case KeyControl.

3.1. Troubleshooting

3.1.1. Secure Boot is not enabled

When attempting to Encrypt a VM, the following error occurs:

```
A general runtime error occurred. Key Provider KeyControl1 is not compatible with the host xx.xxx.xxx.201.
Reason: "Attestation failed because Secure Boot is not enabled."
```

To resolve this, make sure **Secure Boot** is enable at the VM.

4. Python file examples

4.1. pykmip.py

```
#!/usr/bin/python

import os
import sys
import time
import logging
import argparse
import threading
from kmip.pie.client import ProxyKmipClient, enums
from kmip.core.factories import attributes

logger = logging.getLogger()
if len(logger.handlers) == 0:
    formatter = logging.Formatter(
        '%(asctime)s %(levelname)s: %(funcName)s:%(lineno)d: %(message)s')
    handler = logging.handlers.WatchedFileHandler('kmip.log')
    handler.setFormatter(formatter)
    logger.addHandler(handler)
    logger.setLevel(logging.DEBUG)

cert = '%s/vta.pem' % (os.getcwd())
cacert = '%s/cacert.pem' % (os.getcwd())
key = '%s/test.key' % (os.getcwd())
print(cert, cacert)
config = {
    'hostname': '10.194.148.160',
    'port' : '5696',
    'cert' : cert,
    'ca': cacert,
    #'key': key,
    'ssl_version': 'PROTOCOL_SSLv23',
    'kmip_version': enums.KMIPVersion.KMIP_1_2
}
KEYFILE = 'keyfile.txt'
COUNT = 10
ASYM_KEYSIZE=1024  #Possible values: 1024, 2048, 4096
SYM_KEYSIZE=256  #Possible values: 128, 256

class KMIPKEK(object):

    def kmip_create_symkey(self, client=None):
        """
        Create KMIP Asymmetric Key
        """
        try:
            if client is None:
                client = ProxyKmipClient(**config)
                client.open()
            uuid = client.create(enums.CryptographicAlgorithm.AES, SYM_KEYSIZE,
                                 operation_policy_name='default',
                                 cryptographic_usage_mask=[enums.CryptographicUsageMask.ENCRYPT,
                                                           enums.CryptographicUsageMask.ENCRYPT])
            #client.activate(uuid)
        except Exception as exc:
            print(exc)
            sys.exit(1)
        return uuid

    def kmip_create_asymkey(self, client=None):
        """
        Creat KMIP Asymmetric key
        """
```

```
        try:
            if client is None:
                client = ProxyKmipClient(**config)
                client.open()
            uuid1, uuid2 = client.create_key_pair(
                            enums.CryptographicAlgorithm.RSA,
                            ASYM_KEYSIZE,
                            operation_policy_name='default',
                            public_usage_mask=[enums.CryptographicUsageMask.VERIFY],
                            private_usage_mask=[enums.CryptographicUsageMask.SIGN])
            #client.activate(uuid)
        except Exception as exc:
            #print (exc, uuid1, uuid2)
            print (exc)
            sys.exit(1)
        return uuid1, uuid2

    def kmip_fetch(self, uuid, client=None):
        """
        Fetch KMIP object
        """
        try:
            if client is None:
                client = ProxyKmipClient(**config)
                client.open()
            key = client.get(uuid)
            attr = client.get_attributes(uuid, ['Cryptographic Length'])
            print(attr)
        except Exception as exc:
            print (exc, uuid)
            sys.exit(1)
        return str(key)

    def kmip_destroy(self, guid, client=None):
        """
        Delete KMIP object
        """
        try:
            if client is None:
                client = ProxyKmipClient(**config)
                client.open()
            client.destroy(guid)
            print('Delete kmip guid: %s', guid)
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)

    def kmip_locate(self, sym=True, offset=0, count = 10, client=None):
        try:
            if sym:
                attr = enums.ObjectType.SYMMETRIC_KEY
            else:
                attr = enums.ObjectType.PRIVATE_KEY
            print("Here")
            if client is None:
                client = ProxyKmipClient(**config)
                client.open()
            f = attributes.AttributeFactory()
            print("Here1")
            key = client.locate(
                        maximum_items=int(count),
#                        offset_items=int(offset),
                        attributes=[
                            f.create_attribute(
                                'x-NETAPP-ClusterId',
                                '50b7a51c-6f12-11e4-88de-123478563412'
                            ),
                            f.create_attribute(
                                'x-NETAPP-KeyType',
                                'AES'
```

```
                        ),
                        f.create_attribute(
                            'x-NETAPP-Product',
                            'Data ONTAP'
                        ),
                        f.create_attribute(
                            'x-NETAPP-VserverId',
                            '46'
                        ),
                    ]
                )
        except Exception as exc:
            print (exc)
            sys.exit(1)
        print(key, len(key))

    def kmip_create_symkey500(self):
        """
        Create and write 500 Symmetric KMIP keys
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            with open(KEYFILE, 'a') as f:
                for i in range(COUNT):
                    uuid = self.kmip_create_symkey(client)
                    key = self.kmip_fetch(uuid, client)
                    print ("%s: UUID:%s Key:%s" % (i, uuid, key))
                    f.write('%s %s\n' % (uuid, key))
        except Exception as exc:
            print (exc, uuid)
            sys.exit(1)

    def kmip_create_asymkey500(self):
        """
        Create and write 500 Symmetric KMIP keys
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            with open(KEYFILE, 'a') as f:
                for i in range(COUNT):
                    uuid1, uuid2 = self.kmip_create_asymkey(client)
                    key1 = self.kmip_fetch(uuid1, client)
                    key2 = self.kmip_fetch(uuid2, client)
                    print ("%s: UUID:%s Key:%s\n" % (i, uuid1, key1))
                    print ("%s: UUID:%s Key:%s\n" % (i, uuid2, key2))
                    f.write('%s %s\n' % (uuid1, key1))
                    f.write('%s %s\n' % (uuid2, key2))
        except Exception as exc:
            print (exc, uuid1, uuid2)
            sys.exit(1)

    def kmip_fetch500(self):
        """
        Fetch and verify KMIP keys against value in file
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            i = 0
            with open(KEYFILE, 'r') as fp:
                for line in fp:
                    arr = line.strip().split()
                    guid = arr[0]
                    key = str(client.get(guid))
                    if key == arr[1]:
                        print ("%s: Key match for GUID: %s" % (i, arr[0]))
                    else:
                        print ("%s: Invalid key. Key from KC : %s Key from file: %s" % (i, key, arr[1]))
                        raise RuntimeError('Invalid key for guid: %s' % guid)
```

```
                i += 1
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)

    def kmip_activate500(self):
        """
        Activate all keys
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            i = 0
            with open(KEYFILE, 'r') as fp:
                for line in fp:
                    arr = line.strip().split()
                    guid = arr[0]
                    client.activate(guid)
                    print ("%s: Activated KMIP Guid: %s" % (i, guid))
                    i += 1
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)

    def kmip_activate(self, guid):
        """
        Activate specific guid
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            client.activate(guid)
            print ("Activated KMIP Guid: %s" % (guid))
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)

    def kmip_revoke(self, guid):
        """
        Activate specific guid
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            client.revoke(enums.RevocationReasonCode.CESSATION_OF_OPERATION,
                          guid)
            print ("Revoked KMIP Guid: %s" % (guid))
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)

    def kmip_revoke500(self):
        """
        Revoke all keys
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            i = 0
            with open(KEYFILE, 'r') as fp:
                for line in fp:
                    arr = line.strip().split()
                    guid = arr[0]
                    client.revoke(enums.RevocationReasonCode.CESSATION_OF_OPERATION,
                                  guid)
                    print ("%s: Revoked KMIP Guid: %s" % (i, guid))
                    i += 1
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)
```

```
    def kmip_destroy500(self):
        """
        Destroy all keys
        """
        try:
            client = ProxyKmipClient(**config)
            client.open()
            i = 0
            with open(KEYFILE, 'r') as fp:
                for line in fp:
                    arr = line.strip().split()
                    guid = arr[0]
                    client.destroy(guid)
                    print ("%s: Deleted KMIP Guid: %s" % (i, guid))
                    i += 1
            os.unlink(KEYFILE)
        except Exception as exc:
            print (exc, guid)
            sys.exit(1)

    def __init__(self):
        parser = argparse.ArgumentParser(
                    description='KMIP commands',
                     usage='''kmip1.py <command> [<args>]
        The commands are:
                create_symkey       Create 256 bit Symmetric KMIP Key and return its guid
                create_asymkey      Create 256 bit Asymmetric KMIP Key and return its guids
                fetch               Fetch Key bits for specified guid
                activate            Activate KMIP object
                revoke              Revoke KMIP object
                destroy             Delete specific key
                kcinfo              Display KeyControl information
                create_symkey500    Create 500 Symmetric KMIP keys and write uuid and KMIP data to a file
                create_asymkey500   Create 500 Asymmetric KMIP keys and write uuid and KMIP data to a file
                verify500           Fetch key from KC and compare it against value in file
                activate500         Activate all keys
                revoke500           Revoke all 500 keys
                destroy500          Destroy all 500 keys
                sym_locate <offset> <count>  Locate symmetric key objects
                asym_locate <offset> <count> Locate asymmetric key objects
              ''')
        parser.add_argument('command', help='Subcommand to run')
        args = parser.parse_args(sys.argv[1:2])
        if not hasattr(self, args.command):
            print ('Unrecognized command: %s' % (args.command))
            parser.print_help()
            exit(1)
        getattr(self, args.command)()

    def create_symkey(self):
        """
        Create KMIP object
        """
        uuid = self.kmip_create_symkey()
        print ("UUID: %s" % (uuid))

    def create_asymkey(self):
        """
        Create Asymmetric KMIP key
        """
        uuid1, uuid2 = self.kmip_create_asymkey()
        print ("UUID1: %s\nUUID2: %s" % (uuid1, uuid2))

    def fetch(self):
        """
        Fetch KMIP object
        """
        parser = argparse.ArgumentParser(
                    description='Fetch KMIP Key object for specified guid')
        parser.add_argument('uuid')
        args = parser.parse_args(sys.argv[2:])
```

```
        key = self.kmip_fetch(args.uuid)
        print ("Key: %s" % (key))

    def destroy(self):
        """
        Destroy kmip object
        """
        parser = argparse.ArgumentParser(
                    description='Delete KMIP Key object for specified guid')
        parser.add_argument('uuid')
        args = parser.parse_args(sys.argv[2:])
        self.kmip_destroy(args.uuid)

    def sym_locate(self):
        parser = argparse.ArgumentParser(
                    description='Skip offset items and return result')
        parser.add_argument('offset')
        parser.add_argument('count')
        args = parser.parse_args(sys.argv[2:])
        self.kmip_locate(True, args.offset, args.count)

    def asym_locate(self):
        parser = argparse.ArgumentParser(
                    description='Skip offset items and return result')
        parser.add_argument('offset')
        parser.add_argument('count')
        args = parser.parse_args(sys.argv[2:])
        print(args.count)
        self.kmip_locate(False, args.offset, args.count)

    def activate(self):
        """
        Activate all keys
        """
        parser = argparse.ArgumentParser(
                    description='Activate a KMIP Guid')
        parser.add_argument('guid')
        args = parser.parse_args(sys.argv[2:])
        self.kmip_activate(args.guid)

    def revoke(self):
        """
        Activate all keys
        """
        parser = argparse.ArgumentParser(
                    description='Revoke a KMIP Guid')
        parser.add_argument('guid')
        args = parser.parse_args(sys.argv[2:])
        self.kmip_revoke(args.guid)

    def activate500(self):
        """
        Activate all keys
        """
        self.kmip_activate500()

    def create_symkey500(self):
        """
        Create 500 KMIP keys and write uuid and KMIP data to a file
        """
        self.kmip_create_symkey500()

    def create_asymkey500(self):
        """
        Create 500 Asymmmetric keys
        """
        self.kmip_create_asymkey500()

    def verify500(self):
        """
        Fetch KMIP keys and verify it against the value in file
```

```
        """
        self.kmip_fetch500()

    def revoke500(self):
        """
        Revoke all keys
        """
        self.kmip_revoke500()

    def destroy500(self):
        """
        Destroy all keys
        """
        self.kmip_destroy500()

    def kcinfo(self):
        """
        Fetch KC information
        """
        print ("KeyControl IP: %s" % (config['hostname']))
        print ("KeyControl KMIP port: %s" % (config['port']))

if __name__ == '__main__':
    KMIPKEK()
```