THALES

Kubernetes Secrets Encryption

INTEGRATION GUIDE
THALES LUNA HSM

Document Information

Document Part Number	007-000790-001
Revision	A
Release Date	22 October 2020

Trademarks, Copyrights, and Third-Party Software

CONTENTS

Overview 4
Certified Platforms 4
Prerequisites 5
  Configure Luna HSM 5
  Set up Kubernetes Cluster 6
Configuring Luna HSM with KMS Plugin for Kubernetes Secret Encryption 7
  Configuring Luna HSM with K8S-KMS-Plugin 7
  Deploying K8S-KMS-Plugin as KMS Provider 13
  Verifying Kubernetes Secret Encryption using KMS Provider 15
Contacting Customer Support 19
  Customer Support Portal 19
  Telephone Support 19
  Email Support 19

Overview

Kubernetes Secrets contain sensitive data like your passwords, keys, and certificates. Kubernetes developed the feature to use KMS encryption provider for Encrypting Secret Data at Rest. The KMS encryption provider uses an envelope encryption scheme to encrypt data in etcd. The data is encrypted using a data encryption key (DEK); a new DEK is generated for each encryption. The DEKs are encrypted with a key encryption key (KEK) that is stored and managed in a remote KMS. The KMS provider uses gRPC to communicate with a specific KMS plugin. The KMS plugin, which is implemented as a gRPC server and deployed on the same host(s) as the Kubernetes master(s), is responsible for all communication with the remote KMS.

Thales has developed a KMS plugin that communicates with a remote KMS for managing Secret Data Encryption where:

- KMS Plugin - K8S-KMS-Plugin
- Remote KMS - Thales Luna HSM

Following are some of the benefits of using Luna HSM along with K8S-KMS-Plugin to generate encryption keys that protect secret data for Kubernetes Secret encryption:

- Secure generation, storage, and protection of encryption keys on FIPS 140-2 level 3 validated hardware
- Full life cycle management of keys
- HSM audit trail
- Significant performance improvements by off-loading cryptographic operations from servers
- Using Cloud services with confidence

Certified Platforms

The following platforms are certified on Thales Luna HSM for this integration:

Kubernetes	KMS Plugin	Operating System
Kubernetes version 1.19.0	K8S-KMS-Plugin	CentOS 7

Thales Luna HSM: Thales Luna HSM appliances are purposefully designed to provide a balance of security, high performance, and usability that makes them an ideal choice for enterprise, financial, and government organizations. Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing. The Thales Luna HSM on premise offerings include the Luna Network HSM, PCIe HSM, and Luna USB HSMs. Luna HSMs are also available for access as an offering from cloud service providers such as IBM cloud HSM and AWS cloud HSM classic.

NOTE: All Luna HSMs support this integration, provided a supported Luna Client is used.

Prerequisites

Complete the following tasks before you proceed with this integration:

Configure Luna HSM

If you are using a Luna HSM:

1. Ensure that the HSM is set up, initialized, provisioned, and ready for deployment. Refer to the Luna HSM Product Documentation for more information.
2. Create a partition on the Luna HSM for use with Kubernetes.
3. Register a client for the system and assign the client to each partition to create an NTLS connection for the three partitions, if you are using a Luna Network HSM. Initialize the Crypto Officer and Crypto User roles for each registered partition.
4. Ensure that each partition is successfully registered and configured. The command to see the registered partitions is:

   ```
   # /usr/safenet/lunaclient/bin/lunacm
   ```

```
[root@k8s-master ~]# /usr/safenet/lunaclient/bin/lunacm
lunacm (64-bit) v10.2.0-111. Copyright (c) 2020 SafeNet. All rights reserved.


        Available HSMs:

        Slot Id ->              0
        Label ->                KubeKMS01
        Serial Number ->        1238696044948
        Model ->                LunaSA 7.4.0
        Firmware Version ->     7.4.0
        Configuration ->        Luna User Partition With SO (PW) Key Export With Cloning Mode
        Slot Description ->     Net Token Slot
        FM HW Status ->         Non-FM

        Slot Id ->              1
        Label ->                KubeKMS02
        Serial Number ->        1238696044908
        Model ->                LunaSA 7.4.0
        Firmware Version ->     7.4.0
        Configuration ->        Luna User Partition With SO (PW) Signing With Cloning Mode
        Slot Description ->     Net Token Slot
        FM HW Status ->         Non-FM

        Slot Id ->              8
        HSM Label ->            KubeKMS-HA
        HSM Serial Number ->    11238696044948
        HSM Model ->            LunaVirtual
        HSM Firmware Version -> 7.4.0
        HSM Configuration ->    Luna Virtual HSM (PW) Signing With Cloning Mode
        HSM Status ->           N/A - HA Group
        HSM Certificates ->     *** Test Certs ***



        Current Slot Id: 0
```

5. Enable partition policies 22 and 23 to allow activation and auto-activation for PED-authenticated HSM.

> **NOTE:** Follow the Luna Network HSM documentation for detailed steps for creating NTLS connection, initializing the partitions, and various user roles. The screenshot above is showing the 2 Luna partitions are configured in HA mode.

Configuring Luna HSM HA (High-Availability)

Please refer to the Luna Network HSM documentation for HA steps and details regarding configuring and setting up two or more HSM appliances on Windows and UNIX systems. You must enable the HAOnly setting in HA for failover to work so that if the primary device stop functioning for some reason, all calls automatically routed to the secondary device till the primary device starts functioning again.

> **NOTE:** This integration is tested in both HA and FIPS mode.

Set up Kubernetes Cluster

Refer to the *Kubernetes Documentation* for installing and running the Kubernetes Cluster. For demonstration, Kubernetes Cluster used in this documentation is setup with **1 Master** and **2 Worker** nodes on VMware. After installation, ensure that Kubernetes Cluster is up and running successfully.

```
[root@k8s-master ~]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-c9784d67d-r9gs5       1/1     Running   6          37d
kube-system   calico-node-fgf7k                             1/1     Running   20         37d
kube-system   calico-node-r8g5c                             1/1     Running   6          37d
kube-system   calico-node-wd5p2                             1/1     Running   14         37d
kube-system   coredns-f9fd979d6-782jh                       1/1     Running   6          37d
kube-system   coredns-f9fd979d6-mxw7s                       1/1     Running   6          37d
kube-system   etcd-k8s-master.kube.com                      1/1     Running   7          37d
kube-system   kube-apiserver-k8s-master.kube.com            1/1     Running   7          37d
kube-system   kube-controller-manager-k8s-master.kube.com   1/1     Running   6          37d
kube-system   kube-proxy-f795r                              1/1     Running   6          37d
kube-system   kube-proxy-fngnz                              1/1     Running   1          37d
kube-system   kube-proxy-q7zqx                              1/1     Running   3          37d
kube-system   kube-scheduler-k8s-master.kube.com            1/1     Running   6          37d
[root@k8s-master ~]# kubectl get nodes
NAME                 STATUS   ROLES    AGE   VERSION
k8s-master.kube.com   Ready    master   37d   v1.19.0
k8s-node1.kube.com    Ready    <none>   37d   v1.19.0
k8s-node2.kube.com    Ready    <none>   37d   v1.19.0
```

Configuring Luna HSM with KMS Plugin for Kubernetes Secret Encryption

Kubernetes Secret encryption uses KMS provider for encryption/decryption request and KMS provider calls the K8S-KMS-Plugin to communicate with Luna HSM. Complete the following tasks on Kubernetes Master. Any configuration updates to the Kubernetes Master will automatically deploy on all Nodes connected to the Master.

> Configuring Luna HSM with K8S-KMS-Plugin

> Deploying K8S-KMS-Plugin as KMS Provider

> Verifying Kubernetes Secret Encryption using KMS Provider

Configuring Luna HSM with K8S-KMS-Plugin

Before starting ensure that Luna Client is installed and NTL Service is configured with the Luna HSM partition on Master Host of Kubernetes Cluster. To configure Luna HSM with K8S-KMS-Plugin:

1. Connect to the master host as root or as a user with administrative privileges.
2. Set the **GOPATH**. Typically, the value will be a directory tree child of your development workspace.

   ```
   # export KMSPATH=/opt/kms
   # mkdir -p $KMSPATH/src/github.com/thalescpl-io
   # cd $KMSPATH/src/github.com/thalescpl-io
   ```

3. Clone the **k8s-kms-plugin** repository in current directory by executing the following command:

   ```
   # git clone https://github.com/thalescpl-io/k8s-kms-plugin.git
   ```

4. Copy the Luna minimal client package at the **k8s-kms-plugin** directory:

   ```
   # cp /home/LunaClient-Minimal-10.2.0-111.x86_64.tar k8s-kms-plugin/
   ```

5. Copy the Luna Configuration file **/etc/Chrystoki.conf** file to **k8s-kms-plugin** directory:

   ```
   # cp /etc/Chrystoki.conf k8s-kms-plugin/
   ```

6. Edit the library path and change the **Chrystoki2** and **Secure Trusted Channel** sections in **k8s-kms-plugin/Chrystoki.conf** file:

   ```
   Chrystoki2 = {
      LibUNIX = /usr/safenet/lunaclient/libs/64/libCryptoki2.so;
      LibUNIX64 = /usr/safenet/lunaclient/libs/64/libCryptoki2_64.so;
   }
   Secure Trusted Channel = {
      ClientTokenLib = /usr/safenet/lunaclient/libs/64/libSoftToken.so;
   }
   ```

7. Copy the certificates and keys required to connect to Luna HSM in **k8s-kms-plugin** directory:

   ```
   # cp -r /usr/safenet/lunaclient/cert/server k8s-kms-plugin/
   # cp -r /usr/safenet/lunaclient/cert/client k8s-kms-plugin/
   ```

8. Create a new file **start.sh** under **k8s-kms-plugin** directory with the minimum required flags to work with Luna HSM.

```
# cd k8s-kms-plugin/
# cat start.sh
```

```
#!/usr/bin/env bash
/k8s-kms-plugin serve --provider luna --p11-lib
/usr/safenet/lunaclient/libs/64/libCryptoki2_64.so --p11-label KubeKMS-HA --
p11-pin userpin1
```

Where **p11-label** and **p11-pin** are partition name and password respectively. Below are the further options for k8s-kms-plugin which you can specify as per your requirements:

```
Usage:
   serve [flags]
Flags:
   --allow-any                Allow any device (accepts all ids/secrets)
   --disable-socket           Disable socket based server
   --enable-server            Enable TLS based server
   -h, --help                 help for serve
   --socket string            Unix Socket (default "/tmp/run/hsm-plugin-server.sock")
   --tls-ca string            TLS CA cert (default "certs/ca.crt")
   --tls-certificate string   TLS server cert (default "certs/tls.crt")
   --tls-key string           TLS server key (default "certs/tls.key")
Global Flags:
   --auto-create              Auto create the keys if needed (default true)
   --ca-id string             Cert ID for CA Cert record (default "1c3d30d5-dfa8-4167-
   a9f9-2c768464181b")
   --config string            ConfigFile)
   --host string              TCP Host (default "0.0.0.0")
   --kek-id string            Key ID for KMS KEK (default "a37807cd-6d1a-4d75-813a-
   e120f30176f7")
   -p, --native-path string   Path to key store for native provider(Files only) (default
   ".keys")
   --output string            Log output format... text or json supported (default "text")
   --p11-key-label string     Key Label to use for encrypt/decrypt (default "k8s-dek")
   --p11-label string         P11 token label
   --p11-lib string           Path to p11 library/client
   --p11-pin string           P11 Pin
   --p11-slot int             P11 token slot
   --port int                 TCP Port for gRPC service (default 31400)
   --provider string          Provider (default "p11")
```

9. Create a new **Dockerfile** or edit the already available **Dockerfile** under **k8s-kms-plugin** directory that will create a docker image containing **k8s-kms-plugin** and all required resources for Luna HSM Client to communicate with the Luna HSM partition.

Ensure that you are providing the correct file name and path for all required resources which are copying from host to docker image.

```
# cat Dockerfile
```

```
## Build Stage
FROM goboring/golang:1.14.6b4 as build
WORKDIR /app
ADD go.mod /app/go.mod
ADD go.sum /app/go.sum
ADD tools.go /app/pkg/tools.go
ADD vendor /app/vendor
ADD pkg /app/pkg
ADD apis /app/apis
ADD cmd/ /app/cmd/

ENV GOOS linux
ENV GOARCH amd64
ENV CGO_ENABLED 1
ENV GOFLAGS -mod=vendor
RUN go build -o k8s-kms-plugin ./cmd/k8s-kms-plugin

### Plugin Server
FROM goboring/golang:1.14.6b4 as kms-server
WORKDIR /
COPY --from=build /app/k8s-kms-plugin /k8s-kms-plugin
COPY LunaClient-Minimal-10.2.0-111.x86_64.tar /tmp/
RUN mkdir -p /usr/safenet/lunaclient
RUN mkdir -p /usr/safenet/lunaclient/cert
RUN mkdir -p /usr/safenet/lunaclient/cert/client
RUN mkdir -p /usr/safenet/lunaclient/cert/server
RUN tar -xvf /tmp/LunaClient-Minimal-10.2.0-111.x86_64.tar --strip 1 -C
/usr/safenet/lunaclient
RUN cp /usr/safenet/lunaclient/openssl.cnf /usr/safenet/lunaclient/bin

ENV ChrystokiConfigurationPath=/etc
COPY Chrystoki.conf /etc/Chrystoki.conf
```

```
COPY server/CAFile.pem /usr/safenet/lunaclient/cert/server
COPY client/k8s-master.kube.comKey.pem /usr/safenet/lunaclient/cert/client
COPY client/k8s-master.kube.com.pem /usr/safenet/lunaclient/cert/client
COPY start.sh /start.sh
RUN chmod +x /start.sh

ENTRYPOINT ["/start.sh"]
```

10. Create an image with `docker build` command and **Dockerfile**:

```
# docker build . -t kms-server
```

You will see a confirmation message similar to the following when the image is built successfully.

```
 ---> 6588f2daec16
Step 26/32 : COPY Chrystoki.conf /etc/Chrystoki.conf
 ---> 458a564e16bf
Step 27/32 : COPY server/CAFile.pem /usr/safenet/lunaclient/cert/server
 ---> ffea4fd02c18
Step 28/32 : COPY client/k8s-master.kube.comKey.pem /usr/safenet/lunaclient/cert/client
 ---> 2a8957309d18
Step 29/32 : COPY client/k8s-master.kube.com.pem /usr/safenet/lunaclient/cert/client
 ---> a9ba6300cb50
Step 30/32 : COPY start.sh /start.sh
 ---> 2efd8eb7e289
Step 31/32 : RUN chmod +x /start.sh
 ---> Running in b16f3424a2c8
Removing intermediate container b16f3424a2c8
 ---> 77d0d858520e
Step 32/32 : ENTRYPOINT ["/start.sh"]
 ---> Running in 8249ed7dbf86
Removing intermediate container 8249ed7dbf86
 ---> cb2a63c48703
Successfully built cb2a63c48703
Successfully tagged kms-server:latest
[root@k8s-master k8s-kms-plugin]#
```

11. The image **kms-server** will be listed along with other images:

```
# docker images
```

The created image will be listed along with other images. The output will be similar to the following:

```
[root@k8s-master k8s-kms-plugin]# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
kms-server                                latest              cb2a63c48703        2 minutes ago       941MB
<none>                                    <none>              47dcb4fba963        2 minutes ago       925MB
k8s.gcr.io/kube-proxy                     v1.19.1             33c60812eab8        6 weeks ago         118MB
k8s.gcr.io/kube-apiserver                 v1.19.1             ce0df89806bb        6 weeks ago         119MB
k8s.gcr.io/kube-controller-manager        v1.19.1             538929063f23        6 weeks ago         111MB
k8s.gcr.io/kube-scheduler                 v1.19.1             49eb8a235d05        6 weeks ago         45.7MB
calico/node                               v3.16.1             0f351f210d5e        6 weeks ago         164MB
calico/pod2daemon-flexvol                 v3.16.1             4cbe1ed86c35        6 weeks ago         22.9MB
calico/cni                                v3.16.1             4ab373b1fac4        6 weeks ago         133MB
calico/kube-controllers                   v3.16.1             03feeb39a75a        6 weeks ago         52.9MB
goboring/golang                           1.14.6b4            30748d05537e        3 months ago        847MB
k8s.gcr.io/etcd                           3.4.9-1             d4ca8726196c        3 months ago        253MB
k8s.gcr.io/coredns                        1.7.0               bfe3a36ebd25        4 months ago        45.2MB
k8s.gcr.io/pause                          3.2                 80d28bedfe5d        8 months ago        683kB
[root@k8s-master k8s-kms-plugin]#
```

12. Create the pod manifest yaml file **kms-plugin.yaml** and ensure that the file has the following contents:

```
# cat kms-plugin.yaml
```

```
apiVersion: v1
kind: Pod

metadata:
  name: k8s-kms-plugin-server
  labels:
    app.kubernetes.io/name: k8s-kms-plugin-server
spec:
  containers:
    - name: plugin-server
      image: "kms-server"
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - mountPath: /certs/
          name: certstore
        - mountPath: /tmp/run
          name: socket
  hostNetwork: true
  priorityClassName: system-node-critical
  volumes:
    - name: certstore
      emptyDir: {}
    - name: ca
      hostPath:
        path: /etc/
    - name: socket
      hostPath:
        path: /tmp/run
        type: DirectoryOrCreate
status: {}
```

13. Copy the pod manifest **kms-plugin.yaml** to the kubernetes pod manifest location used by **kubelet** service for running all the static pods.

```
# cp kms-plugin.yaml /etc/kubernetes/manifests/
```

Where "/**etc/kubernetes/manifest**" is the static pod manifest location. Location for pod manifest may vary for cluster deployment so ensure that you are using the correct location.

14. Kubernetes will deploy k8s-kms-plugin pod automatically from pod manifest location. Verify the deployment status:

```
# kubectl get pods
```

```
[root@k8s-master k8s-kms-plugin]# kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
k8s-kms-plugin-server-k8s-master.kube.com   1/1     Running   0          15s
[root@k8s-master k8s-kms-plugin]#
```

15. On Master host, verify that the socket file is created under /**tmp**/**run** directory that you specified in **kms-plugin.yaml**.

```
[root@k8s-master k8s-kms-plugin]# ls -ltr /tmp/run
total 0
srwxrwxr-x 1 root root 0 Oct 22 16:15 hsm-plugin-server.sock
[root@k8s-master k8s-kms-plugin]#
```

16. Connect the pod to verify that k8s-kms-plugin is integrated with Luna HSM and encryption key is generated. Execute the following command on the master host to connect pod:

```
# kubectl exec -it k8s-kms-plugin-server-k8s-master.kube.com -- /bin/bash
```

17. Verify that the pod have access to the Luna HSM partition:

```
# /usr/safenet/lunaclient/bin/64/vtl listslots
```

```
root@k8s-master:/# /usr/safenet/lunaclient/bin/64/vtl listslots
vtl (64-bit) v10.2.0-111. Copyright (c) 2020 SafeNet. All rights reserved.

Number of slots: 1

The following slots were found:

Slot Description          Label                            Serial #         Status
==== ==================== ================================ ================ ============
   0 HA Virtual Card Slot KubeKMS-HA                       11238696044948   Present
root@k8s-master:/#
```

18. Verify the contents of registered Luna HSM Partition to ensure that encryption key is generated on Luna HSM via KMS-Plugin.

```
lunacm:>partition contents

        The User is currently logged in.  Looking for objects in the
        User's partition.

        Object list:

        Label:          k8s-dek
        Handle:         2000001
        Object Type:    Symmetric Key
        Object UID:     2300000010000002d301e0800


        Number of objects:  1


Command Result : No Error
```

KMS plugin is now configured to use the Luna HSM and UNIX socket "**hsm-plugin-server.sock**" is created on Master Node where API Server is running and ready to serve the request from API Server.

Deploying K8S-KMS-Plugin as KMS Provider

Now we will deploy KMS-Plugin as the KMS provider and configure the API Server to use the KMS provider for Kubernetes Secret encryption.

To deploy the K8S-KMS-Plugin as KMS Provider:

1. Create the encryption configuration file **encryption-config.yaml** when the KMS-Plugin is ready to use as an encryption provider on all the same nodes as your API servers:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
    providers:
    - kms:
        name: k8s-kms-plugin
        endpoint: unix:///tmp/run/hsm-plugin-server.sock
        cachesize: 100
        timeout: 3s
    - identity: {}
```

2. Save the encryption configuration file **encryption-config.yaml** at any location on the Master host. For Example:

   `/etc/kubernetes/kms/encryption-config.yaml`

3. Open the **kube-apiserver.yaml** file from static pod manifest path for editing and add the **--encryption-provider-config flag** to the command list of kube-apiserver. The flag must point to the **encryption-config.yaml** file. Here's an example:

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --encryption-provider-config=/etc/kubernetes/kms/encryption-config.yaml
    - --advertise-address=10.164.76.80
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    ...
```

4. For kube-apiserver pod to communicate with the k8s-kms-plugin, access to the directories where **encryption-config.yaml** file and UNIX socket is located are needed. To provide access, add mount points in **kube-apiserver.yaml** as shown below and ensure the correct indentation:

```
   ...
   volumeMounts:
   -  mountPath: /etc/kubernetes/kms
      name: kms
      readOnly: true
   -  mountPath: /tmp/run
      name: socket
   ...
   ...
   volumes:
   - hostPath:
       path: /etc/kubernetes/kms
       type: DirectoryOrCreate
     name: kms
   - hostPath:
       path: /tmp/run
       type: DirectoryOrCreate
     name: socket
...
```

5. Save and close the **kube-apiserver.yaml** file. The API server will be restarted automatically when you save the changes. Ensure that your cluster back online without any failure and API Server is READY and Running.

```
# kubectl get pods --all-namespaces -o wide
```

```
[root@k8s-master k8s-kms-plugin]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY   STATUS    RESTARTS   AGE
default       k8s-kms-plugin-server-k8s-master.kube.com     1/1     Running   0          25m
kube-system   calico-kube-controllers-c9784d67d-r9gs5       1/1     Running   8          37d
kube-system   calico-node-fgf7k                             1/1     Running   20         37d
kube-system   calico-node-r8g5c                             1/1     Running   8          37d
kube-system   calico-node-wd5p2                             1/1     Running   14         37d
kube-system   coredns-f9fd979d6-782jh                       1/1     Running   8          37d
kube-system   coredns-f9fd979d6-mxw7s                       1/1     Running   8          37d
kube-system   etcd-k8s-master.kube.com                      1/1     Running   10         37d
kube-system   kube-apiserver-k8s-master.kube.com            1/1     Running   0          6m47s
kube-system   kube-controller-manager-k8s-master.kube.com   1/1     Running   10         37d
kube-system   kube-proxy-f795r                              1/1     Running   8          37d
kube-system   kube-proxy-fngnz                              1/1     Running   1          37d
kube-system   kube-proxy-q7zqx                              1/1     Running   3          37d
kube-system   kube-scheduler-k8s-master.kube.com            1/1     Running   10         37d
[root@k8s-master k8s-kms-plugin]#
```

6. Optionally, verify the API Server logs to ensure that API server is running with the encryption provider.

```
I1022 20:34:19.689296       1 server.go:163] Version: v1.19.1
I1022 20:34:21.805715       1 clientconn.go:106] parsed scheme: ""
I1022 20:34:21.805755       1 clientconn.go:106] scheme "" not registered, fallback to default scheme
I1022 20:34:21.805813       1 passthrough.go:48] ccResolverWrapper: sending update to cc: {[{/tmp/run/hsm-plugin-server.sock  <nil> 0 <nil>}] <nil> <nil>}
I1022 20:34:21.805847       1 clientconn.go:948] ClientConn switching balancer to "pick_first"
I1022 20:34:21.806933       1 clientconn.go:106] parsed scheme: ""
I1022 20:34:21.806962       1 clientconn.go:106] scheme "" not registered, fallback to default scheme
I1022 20:34:21.806989       1 passthrough.go:48] ccResolverWrapper: sending update to cc: {[{/tmp/run/hsm-plugin-server.sock  <nil> 0 <nil>}] <nil> <nil>}
I1022 20:34:21.807005       1 clientconn.go:948] ClientConn switching balancer to "pick_first"
```

7. If API Server is configured successfully to use the k8s-kms-plugin, the connection will be established and you will see similar to the following when running the below command on Master node:

```
# ss -a --unix -p | grep hsm-plugin-server.sock
```

```
[root@k8s-master k8s-kms-plugin]# ss -a --unix -p | grep hsm-plugin-server.sock
u_str  LISTEN     0      128    /tmp/run/hsm-plugin-server.sock 85460              * 0                  users:(("k8s-kms-plugin",pid=12316,fd=7))
u_str  ESTAB      0      0      /tmp/run/hsm-plugin-server.sock 150601             * 150600             users:(("k8s-kms-plugin",pid=12316,fd=10))
u_str  ESTAB      0      0      /tmp/run/hsm-plugin-server.sock 149866             * 150603             users:(("k8s-kms-plugin",pid=12316,fd=11))
[root@k8s-master k8s-kms-plugin]#
```

API sever is now configured to use k8s-kms-plugin as KMS Provider and k8s-kms-plugin is ready to serve the request from API server for secret encryption and decryption.

Verifying Kubernetes Secret Encryption using KMS Provider

Kubernetes Secret is encrypted when written to etcd. After restarting your kube-apiserver, any newly created or updated secret will be encrypted when stored. To verify, you can use the `etcdctl` command line program to retrieve the contents of your secret. To Verify the Secret Encryption using KMS Provider:

1. Create a new secret called **mysecret** in the default namespace.

```
# kubectl create secret generic mysecret -n default --from-
literal=mykey=mys3cr3t
```

The encrypted secret gets saved in the etcd.

2. Read the secret out of etcd using the etcdctl command line. The command is listed below with parameter values. Ensure to change all parameter values as per your environment.

```
# alias etcdctl3="ETCDCTL_API=3
/var/lib/docker/overlay2/b672106e8ed998f9a4a591175d0d79c5e0d64ecc4d465419aec43a
0458c9daf7/diff/usr/local/bin/etcdctl --endpoints="https://127.0.0.1:2379" --
cert=/etc/kubernetes/pki/apiserver-etcd-client.crt --
key=/etc/kubernetes/pki/apiserver-etcd-client.key --
cacert=/etc/kubernetes/pki/etcd/ca.crt"

# etcdctl3 get /registry/secrets/default/mysecret
```

You will see output similar to the example below:

```
[root@k8s-master k8s-kms-plugin]# etcdctl3 get /registry/secrets/default/mysecret
/registry/secrets/default/mysecret
k8s:enc:kms:v1:k8s-kms-plugin:▒eyJhbGciOiJkaXIiLCJraWQiOiJrOHMtZGVrIiwiZW5jIjoiQTI1NkdDTSJ9.
N▒!~▒▒t▒▒x▒▒▒▒Una▒^▒w▒9D~▒▒ë+Ij▒$T▒▒▒▒▒R▒▒:▒lKV`)▒▒o▒P2▒\'▒▒▒b@?▒-i▒▒ ^O=A7EH▒Cr▒▒▒▒▒▒]▒z
```

3. Run the following command to ensure that stored secret is prefixed with the **k8s:enc:kms:v1:k8s-kms-plugin**, which indicates that the KMS Provider has encrypted the resulting data.

```
# etcdctl3 get /registry/secrets/default/mysecret | hexdump -C
```

```
[root@k8s-master k8s-kms-plugin]# etcdctl3 get /registry/secrets/default/mysecret | hexdump -C
00000000  2f 72 65 67 69 73 74 72  79 2f 73 65 63 72 65 74  |/registry/secret|
00000010  73 2f 64 65 66 61 75 6c  74 2f 6d 79 73 65 63 72  |s/default/mysecr|
00000020  65 74 0a 6b 38 73 3a 65  6e 63 3a 6b 6d 73 3a 76  |et.k8s:enc:kms:v|
00000030  31 3a 6b 38 73 2d 6b 6d  73 2d 70 6c 75 67 69 6e  |1:k8s-kms-plugin|
00000040  3a 00 97 65 79 4a 68 62  47 63 69 4f 69 4a 6b 61  |:..eyJhbGciOiJka|
00000050  58 49 69 4c 43 4a 72 61  57 51 69 4f 69 4a 72 4f  |XIiLCJraWQiOiJrO|
00000060  48 4d 74 5a 47 56 72 49  69 77 69 5a 57 35 6a 49  |HMtZGVrIiwiZW5jI|
00000070  6a 6f 69 51 54 49 31 4e  6b 64 44 54 53 4a 39 2e  |joiQTI1NkdDTSJ9.|
00000080  2e 4f 6f 6c 30 6f 73 55  55 49 55 65 55 46 50 71  |.Ool0osUUIUeUFPq|
00000090  6b 53 58 52 31 41 51 2e  78 4d 50 6c 58 65 49 4b  |kSXR1AQ.xMPlXeIK|
000000a0  35 6b 34 42 2d 78 2d 6e  55 6f 34 49 55 79 4c 67  |5k4B-x-nUo4IUyLg|
000000b0  79 43 37 4a 67 66 48 45  4c 5a 41 38 30 72 76 30  |yC7JgfHELZA80rv0|
000000c0  48 50 45 2e 59 79 6a 45  48 51 52 62 79 6f 63 47  |HPE.YyjEHQRbyocG|
000000d0  49 6e 41 2d 76 6d 79 51  51 41 43 8f f1 5a ae 71  |InA-vmyQQAC..Z.q|
000000e0  bc 58 67 1c 6c 37 3f 87  f8 85 df ed b2 52 84 f1  |.Xg.l7?......R..|
000000f0  16 b3 61 8f ae cc 1e 4c  e3 7a 31 09 81 d7 81 b8  |..a....L.z1.....|
00000100  fc d1 72 c1 99 2d fd a2  dd ae 1d 41 44 1d 70 2d  |..r..-.....AD.p-|
00000110  8d 4d 27 a1 86 54 73 ee  d5 d1 79 ee 9c 8b 37 cf  |.M'..Ts...y...7.|
00000120  05 1d 73 cf 68 f5 aa 9d  06 af f4 82 50 6c e2 8f  |..s.h.......Pl..|
00000130  c5 6e 61 9c 5e c3 0e 77  e4 39 44 7e 92 97 d1 91  |.na.^..w.9D~....|
00000140  2b 49 6a a5 24 19 54 a0  a9 cd e3 10 b0 8e 52 ab  |+Ij.$.T.......R.|
00000150  01 c9 3a ed 6c 4b 56 19  60 29 b2 f9 6f eb 50 32  |..:.lKV.`)..o.P2|
00000160  ea 16 5c 27 e8 12 f1 9a  f5 b4 a8 12 62 40 3f ff  |..\'........b@?.|
00000170  2d 15 c9 a8 dd b0 b5 07  d3 5e 4f 13 3d 41 37 45  |-.........^O.=A7E|
00000180  48 01 80 43 72 88 8f 95  fe a3 01 e4 dc 5d 81 7a  |H..Cr........].z|
00000190  b5 55 ca 83 54 15 ae 5c  c8 a3 40 f7 1e b0 89 e9  |.U..T..\..@.....|
000001a0  13 c5 fc c1 ee ba 86 98  5f 6f 63 1b 1a a8 f7 62  |........_oc....b|
000001b0  6e 6c c5 ca 42 ae 50 23  7d 82 f8 f2 c0 0e 60 20  |nl..B.P#}.....` |
000001c0  5d 95 12 a3 bc 10 0d 4e  b1 21 7e 95 8c ff 1e 74  |].......N.!~....t|
000001d0  f2 bd f4 78 1d ed e4 f9  e1 55 0a                 |...x.....U.|
000001db
```

4. Ensure that the secret is correctly decrypting when retrieved via the API Server.

```
# kubectl describe secret mysecret -n default
```

```
[root@k8s-master k8s-kms-plugin]# kubectl describe secret mysecret -n default
Name:         mysecret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
mykey:  8 bytes
[root@k8s-master k8s-kms-plugin]# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-crxw4   kubernetes.io/service-account-token   3      37d
mysecret              Opaque                                1      6m30s
[root@k8s-master k8s-kms-plugin]#
```

5. Run the following command to encrypt all pre-existing secrets. The command reads all secrets and then encrypt all secrets using KMS Provider.

```
# kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

```
[root@k8s-master k8s-kms-plugin]# kubectl get secrets --all-namespaces -o json | kubectl replace -f -
secret/default-token-crxw4 replaced
secret/mysecret replaced
secret/default-token-8tc88 replaced
secret/default-token-msg7f replaced
secret/attachdetach-controller-token-s4k2r replaced
secret/bootstrap-signer-token-wbkgp replaced
secret/calico-kube-controllers-token-ld7hs replaced
secret/calico-node-token-hxhqx replaced
secret/certificate-controller-token-8rv2m replaced
secret/clusterrole-aggregation-controller-token-6lmh7 replaced
secret/coredns-token-gdqr5 replaced
secret/cronjob-controller-token-vwm8z replaced
secret/daemon-set-controller-token-8rd9h replaced
secret/default-token-ntlqk replaced
secret/deployment-controller-token-xlrw5 replaced
secret/disruption-controller-token-65kq6 replaced
secret/endpoint-controller-token-sk2kx replaced
secret/endpointslice-controller-token-lfnds replaced
secret/endpointslicemirroring-controller-token-dhfmg replaced
secret/expand-controller-token-ml45t replaced
secret/generic-garbage-collector-token-qllzl replaced
secret/horizontal-pod-autoscaler-token-9qcrp replaced
secret/job-controller-token-d5rl9 replaced
secret/kube-proxy-token-xl5zd replaced
secret/namespace-controller-token-5v5tq replaced
secret/node-controller-token-4znvd replaced
secret/persistent-volume-binder-token-sgqlq replaced
secret/pod-garbage-collector-token-ffk7l replaced
secret/pv-protection-controller-token-rdplz replaced
secret/pvc-protection-controller-token-l7j77 replaced
secret/replicaset-controller-token-8g6pj replaced
secret/replication-controller-token-4nd5f replaced
secret/resourcequota-controller-token-6qlh2 replaced
secret/service-account-controller-token-sg7d4 replaced
secret/service-controller-token-59z2j replaced
secret/statefulset-controller-token-pxkpp replaced
secret/token-cleaner-token-d2mgk replaced
secret/ttl-controller-token-mql4l replaced
[root@k8s-master k8s-kms-plugin]#
```

If an error occurs due to a conflicting write, retry the command. For larger clusters, it is recommended to subdivide the secrets by namespace or script an update. All secrets are now encrypted using key generated on Luna HSM via k8s-kms-plugin. The secrets will be decrypted automatically via k8s-kms-plugin when called by API Server.

Switching from a local encryption provider to the KMS provider

If you have enabled the native encryption provider and want to migrate to KMS Provider to enhanced security. To switch from a native encryption provider to the KMS provider and re-encrypt all of the secrets using KMS provider perform the steps to Configuring Luna HSM with K8S-KMS-Plugin:

1. Edit the encryption configuration file **encryption-config.yaml** on the Master host. Add the KMS provider as the first entry in the configuration file, as shown in the example below:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
    providers:
    - kms:
        name: k8s-kms-plugin
        endpoint: unix:///tmp/run/socket.sock
        cachesize: 100
        timeout: 3s
    - aescbc:
        keys:
          - name: key1
            secret: <BASE 64 ENCODED SECRET>
```

   Where `key1` is the name of your key and `<BASE 64 ENCODED SECRET>` is actual key of native encryption provider. Ensure that the Unix Socket path is mounted in Kube-API server manifest yaml.

2. Restart all kube-apiserver processes.
3. Run the following command to force all secrets to be re-encrypted using the KMS Provider

```
# kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

This completes the Kubernetes Secret Encryption using k8s-kms-plugin as KMS Provider for secret encryption and Luna HSM to secure the encryption keys on FIPS-validated hardware security modules and provide a higher level of security than the locally-stored encryption keys.

Contacting Customer Support

If you encounter a problem while installing, registering, or operating this product, refer to the documentation. If you cannot resolve the issue, contact your supplier or Thales Customer Support. Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at https://supportportal.thalesgroup.com, is a database where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable repository of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

> **NOTE:** You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone Support

If you have an urgent problem, or cannot access the Customer Support Portal, you can contact Thales Customer Support by telephone at +1 410-931-7520. Additional local telephone support numbers are listed on the support portal.

Email Support

You can also contact technical support by email at technical.support.DIS@thalesgroup.com.