DATASTAX

DataStax Cassandra TDE and Entrust KeyControl

Integration Guide

2024-11-21

Table of Contents

Chapter 1. Introduction

Entrust KeyControl is a Key Management System (KMS) offers functionalities to create, manage, distribute, and safeguard cryptographic keys. It is deployed as a cluster of virtual appliances that integrate with FIPS 140-2 compliant third-party hardware security modules (HSMs) to securely store keys.

Using DataStax Cassandra TDE along with Entrust KeyControl for key management allows you to protect sensitive data wherever it resides.

1.1. Documents to read first

- Key Control Vault v10.3 Online Documentation Set
- Release notes for DataStax Enterprise 6.9
- Getting started with DataStax Enterprise 6.9
- Configuring KMIP encryption

1.2. Product configuration

Product	Version	Notes
DataStax Cassandra	6.9	
KeyControl or KCaaS KMS	10.3.1	KMIP Vault installed and deployed per your environment

Chapter 2. Procedures

High Level steps for implementing Postgres TDE using Entrust KeyControl:

1. Check the prerequisites.
2. Create a certificate request (CSR) for the DSE Server.
3. Generate a KMIP client certificate for the DSE Server.
4. Test access to the KMIP Service.
5. Add a KMIP Host.
6. Encrypt Table Data.

2.1. Check Prerequisites

Before implementing Postgres TDE using Entrust KeyControl, complete the following tasks:

1. Check the Python 3 installation on the Cassandra DSE server:

```
$ python3 --version Python 3.11.9
```

2. Check the Java installation

```
$ java --version

OpenJDK Runtime Environment (build 21.0.4+7-Ubuntu-1ubuntu224.04)
OpenJDK 64-Bit Server VM (build 21.0.4+7-Ubuntu-1ubuntu224.04, mixed mode, sharing)
```

   If you are using OpenJDK, the results should resemble the following output:

```
openjdk version "11.0.x" YYYY-MM-DD
OpenJDK Runtime Environment 18.9 (build 11.0.x+xx)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.x+xx, mixed mode)
```

3. Check that the Cassandra (DSE) service is running

```
$ systemctl status dse.service
● dse.service - LSB: DataStax Enterprise

     Loaded: loaded (/etc/init.d/dse; generated)
     Active: active (running) since Fri 2024-09-06 21:32:21 UTC; 9s ago
       Docs: man:systemd-sysv-generator(8)
    Process: 864 ExecStart=/etc/init.d/dse start (code=exited, status=0/SUCCESS)
      Tasks: 377 (limit: 9519)
     Memory: 2.7G (peak: 2.7G)
        CPU: 47.372s
     CGroup: /system.slice/dse.service
```

```
            └─1720 /usr/lib/jvm/java-11-openjdk-amd64//bin/java -Ddse.server_process
-Djdk.attach.allowAttachSelf=true --add-exports java.base/jdk.internal.misc=ALLUNNAMED --add-opens
java.base/jdk.internal.module=ALL-UNNAMED --add-exports java.base/jdk.internal.ref=A>
```

4. Launch `cqlsh`

```
$ /usr/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 6.8.0 | DSE 6.9.2 | CQL spec 3.4.5 | DSE protocol v2]
Use HELP for help.
cqlsh>
```

2.2. Create a certificate request (CSR) for the DSE server

1. Change your working directory to the `$home`

```
$ cd ~
```

2. Create a new certificate request with a three-year duration:

```
$ openssl req -new -nodes -keyout dse-serv.key -out dse-serv.csr
Generating a RSA private key
.....................................................................................
..+++++
.....................................................................................
+++++ writing new private key to 'dse-serv.key'

-----

You are about to be asked to enter information that will be incorporated into your
certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value, If you enter '.', the field will be
left blank.

-----

Country Name (2 letter code) [XX]:UK
State or Province Name (full name) []:England
Locality Name (eg, city) [Default City]:London
Organization Name (eg, company) [Default Company Ltd]:Entrust
Organizational Unit Name (eg, section) []:DPS
Common Name (eg, your name or your server's hostname) []:dse-serv
Email Address []:<user_email_address>

Please enter the following 'extra' attributes to be sent with your
certificate request
A challenge password []:<empty>
An optional company name []:Entrust
```

3. Verify that the CSR has been generated:

```
$ ls
dse-serv.csr dse-serv.key
```

4. Download the `csr` file to your local machine:

```
$ scp user@<dse_hostname>:dse-serv.csr . dse-serv.csr
```

The generated file dse-serv.csr will be used in the next step to generate a KMIP client certificate.

2.3. Generate a KMIP Client Certificate for the DSE server

1. Sign into the KCaaS or KeyControl KMIP vault webGUI using an account with Security Admin privileges.
2. Navigate to **Security** > **Client certificates**.
3. Select the **+** button.
4. In the Create Client Certificate dialog box, specify the options you want to use.
   - **Certificate Name**: `dse-serv`
   - **Certificate Expiration**: **3 years from today**
   - **Certificate Signing Request (CSR)**: `dse-serv.csr`
   - **Encrypt Certificate Bundle**: **No**
5. Select **Create**.

A message is dispayed indicating that the certificate has been successfully generated.

6. Select the check box next to the newly generated certificate.
7. Select **Download Certificate**. A zip file is downloaded by the browser.

The zip file contains a client certificate `.pem` file and a CA certificate file called `cacert.pem`.

8. Extract the zip file contents:

```
$ unzip dse-serv_2024-*.zip
Archive: dse-serv_2023-12-15-16-54-42.zip
inflating: dse-serv.pem
inflating: cacert.pem
```

9. Upload the two `.pem` files to the DSE linux machine:

```
$ scp dse-serv.pem user@<dse_hostname>:.
$ scp cacert.pem user@<dse_hostname>:.
```

2.3.1. Test access the KMIP Service

1. Show information about the TLS certificate of the KMIP server (KMS):

```
$ echo | openssl s_client -servername <kmip_hostname> -connect <kmip_hostname>:5696
2>/dev/null | openssl x509 -noout -issuer -subject -dates -ext subjectAltName fingerprint
issuer=C = US, O = HyTrust Inc., CN = HyTrust KeyControl Certificate Authority
subject=C = US, O = HyTrust Inc., CN = ip-172-21-103-172.skynet.entrust.cloud
notBefore=Jun 1 00:00:00 2011 GMT notAfter=Dec 31 23:59:59 2049 GMT
X509v3 Subject Alternative Name:
    IP Address:172.21.103.172, DNS:ip-172-21-103-172, DNS:ip-172-21-103-172.skynet.entrust.cloud
SHA1 Fingerprint=30:10:0E:9D:DA:2B:24:8C:B1:96:DE:1F:17:08:20:59:51:60:94:04
```

2. Test the CA certificate:

```
$ openssl s_client -CAfile cacert.pem -connect <kmip_hostname>:5696 Client Certificate Types: RSA sign, DSA
sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+S
HA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms:
RSA+SHA512:ECDSA+SHA512:RSA+SHA384:ECDSA+SHA384:RSA+SHA256:ECDSA+SHA256:RSA+SHA224:ECD
SA+SHA224:RSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA256
Peer signature type: RSA
Server Temp Key: ECDH, P-256, 256 bits
--
SSL handshake has read 2813 bytes and written 460 bytes
Verification: OK
--
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE No
ALPN negotiated
SSL-Session:
 Protocol : TLSv1.2
 Cipher : ECDHE-RSA-AES256-GCM-SHA384
Session-ID: 2ABAF7C3DF81A077634ED5E8480DA23C4D9C1CD3A0C46982224F4DA38B93CE73
Session-ID-ctx: Master-Key:
20606FADDD52929948B0D754D19F412FA032ACB4859D7E5B0B689BFCCB2BE73D5566EA58F8EA5837E168210046FA5AE4
PSK identity: None
PSK identity hint: None
SRP azureusername: None
TLS session ticket lifetime hint: 300 (seconds)
TLS session ticket:
0000 - c2 d0 4a 51 b8 fb fe 66-4f e8 42 3c fe 85 66 3e ..JQ...fO.B<..f>
0010 - 8b 17 a0 f3 9b fa e8 e0-92 b9 98 87 f4 ca 8f 98 ................
0020 - 7a 46 55 49 dd 7e 01 62-e9 1b cd 26 44 7f d3 10 zFUI.~.b...&D...
0030 - b7 77 f3 c9 bb 1b 4a be-82 c2 4f 43 d8 f2 8d 18 .w....J...OC....
0040 - f1 62 5f 9a 55 02 ec e7-b6 76 17 41 46 7d d3 c8 .b_.U....v.AF}..
0050 - e0 62 7b f2 7b ee ba 3d-49 1a 15 bb 76 ed 7a b0
.b{.{..=I...v.z. 0060 - 70 76 bc c0 20 b6 a9 83-86 f4 8d 16 e2 4e 58
```

```
7c pv.. ........NX| 0070 - 75 83 7c d6 67 e4 16 1f-d4 94 9d e9 b3 9c ba
96 u.|.g.......... 0080 - 79 8c d5 52 3d 5d 04 72-61 aa c9 29 06 38 45
4f y..R=].ra..).8EO 0090 - d8 03 86 fc 5f 26 56 d8-c6 cf 74 88 77 2d 00
ab ...._&V...t.w-.. 00a0 - 7a 49 6b 21 97 c3 9d 54-fc 8f 76 d6 be fa 08
36 zIk!...T..v....6 00b0 - c2 57 c7 96 83 ff 54 c2-0c 70 b3 c6 c1 e5 00
a1 .W....T..p......
Start Time: 1563985354
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
-- closed
```

2.4. Add a KMIP host

1. Copy the certificate and CA `.pem` files to the home directory of the DSE server.
2. Convert the key pair from PEM to a DSE compatible JKS format.

```
$ openssl pkcs12 -export -out dse-serv.p12 -inkey dse-serv.key -in dse-serv.pem
Enter Export Password:<password>
Verifying - Enter Export Password:<password>
```

3. Create a JKS keystore:

```
$ keytool -importkeystore -destkeystore dse-serv.jks -srcstoretype PKCS12 -srckeystore dse-serv.p12
Importing keystore dse-serv.p12 to dse-serv.jks...
Enter destination keystore password:<password>
Re-enter new password:<password>
Enter source keystore password:<password>
Entry for alias 1 successfully imported.
Import command completed: 1 entries successfully imported, 0 entries failed or cancelled
```

4. Use openssl to extract the both the KMIP server certificate and the KMIP CA certificate:

```
$ openssl s_client -connect <kmip_hostname>:5696 -showcerts
```

This returns the entire certificate chain, including the KMIP server certificate and possibly the intermediate and CA certificates.

The KMIP Server Certificate is the server's own certificate that the client (your JVM) is trying to validate.

The CA Certificate ensures that the JVM trusts authority that issued the server certificate, creating a valid certificate chain. Without the CA certificate, you may get errors related to an incomplete certification path.

5. Copy each certificate (from `-----BEGIN CERTIFICATE-----` to `-----END CERTIFICATE-----`) and save them as separate `.crt` files, for example:
   - KMIP server certificate: `kmip-server-cert.crt`
   - CA certificate, if available: `ca-cert.crt`
6. Import the KMIP Server Certificate into the Truststore:

```
$ sudo keytool -import -alias kmip-server-cert -file
kmip-server-cert.crt -keystore kmip_truststore.jks -storepass <password>
Certificate was added to keystore
```

7. Verify that the KMIP server certificate has been added correctly:

```
$ keytool -list -v -keystore /etc/dse/conf/kmip_truststore.jks -storepass <password> | grep -A 1 kmip-
server-cert
Alias name: kmip-server-cert
Creation date: Sep 6, 2024
```

8. Verify that the KMIP CA certificate has been added correctly:

```
$ keytool -list -v -keystore /etc/dse/conf/kmip_truststore.jks -storepass <password> | grep -A 1 kmip-
server-cert
Alias name: kmip-server-cert
Creation date: Sep 6, 2024
```

9. Move the keystore and truststore to a directory accessible by DSE and change the file to allow the DSE account read/write access:

```
$ sudo mkdir /etc/dse/conf
$ sudo chown cassandra:cassandra /etc/dse/conf/
$ sudo cp *.jks /etc/dse/conf
$ sudo chown cassandra:cassandra /etc/dse/conf/*.*
```

10. Delete or secure the files used to create the keystore and truststore.
11. Add the host details to the `kmip_hosts` section of the `/etc/dse/dse.yaml`:

```
kmip_hosts:
  <kmip_group_name>:
    hosts: <kmip_hostname>
    keystore_path: /etc/dse/conf/dse-serv.jks
    keystore_type: jks
    keystore_password: <password>
    truststore_path: /etc/dse/conf/kmip_truststore.jks
    truststore_type: jks
    truststore_password: <password>
    key_cache_millis: 300000
    timeout: 1000
    protocol: <protocol> cipher_suites: <supported_cipher>
```

12. Restart DataStax Enterprise:

```
$ sudo service dse restart
```

13. Check that the DSE service is running:

```
$ systemctl status dse.service
● dse.service - LSB: DataStax Enterprise

     Loaded: loaded (/etc/init.d/dse; generated)
     Active: active (running) since Fri 2024-09-06 21:32:21 UTC; 9s ago
       Docs: man:systemd-sysv-generator(8)
    Process: 864 ExecStart=/etc/init.d/dse start (code=exited, status=0/SUCCESS)
      Tasks: 377 (limit: 9519)
     Memory: 2.7G (peak: 2.7G)
        CPU: 47.372s
     CGroup: /system.slice/dse.service
             └─1720 /usr/lib/jvm/java-11-openjdk-amd64//bin/java -Ddse.server_process
-Djdk.attach.allowAttachSelf=true --add-exports java.base/jdk.internal.misc=ALLUNNAMED --add-opens
java.base/jdk.internal.module=ALL-UNNAMED --add-exports java.base/jdk.internal.ref=A>
```

2.5. Encrypt table data

1. Back up the configuration files:

```
$ sudo cp /etc/dse/dse.yaml /etc/dse/dse.yaml.save
$ sudo cp /etc/dse/cassandra/cassandra.yaml
/etc/dse/cassandra/cassandra.yaml.save
```

2. Run `cqlsh`:

```
$ /usr/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 6.8.0 | DSE 6.9.2 | CQL spec 3.4.5 | DSE protocol v2]
Use HELP for help.
cqlsh>
```

3. Create a keyspace for your application:

```
> CREATE KEYSPACE my_keyspace
WITH replication = {'class': 'SimpleStrategy', 'replication_factor':
3};
```

4. After creating your keyspace, you can use it with:

```
> USE my_keyspace;
```

5. Create an encrypted table:

```
> CREATE TABLE my_keyspace.customers (
    customer_id UUID PRIMARY KEY, first_name text, last_name text, email text
)
WITH COMPRESSION = {
    'class': 'Encryptor',
    'key_provider': 'KmipKeyProviderFactory',
    'kmip_host': 'kmip.entrust.com',
    'cipher_algorithm': 'AES/ECB/PKCS5Padding',
    'secret_key_strength': 128
};
```

6. Describe the created database:

```
> USE my_keyspace;
> DESCRIBE TABLE customers;
CREATE TABLE my_keyspace.customers (
    customer_id uuid PRIMARY KEY, email text, first_name text, last_name text
)
WITH additional_write_policy = '99PERCENTILE'
AND bloom_filter_fp_chance = 0.01
AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
AND compression = {'chunk_length_in_kb': '64', 'cipher_algorithm': 'AES/ECB/PKCS5Padding', 'class':
'org.apache.cassandra.io.compress.Encryptor', 'key_provider': 'KmipKeyProviderFactory', 'kmip_host':
'kmip.entrust.com', 'secret_key_strength': '128'}
AND crc_check_chance = 1.0
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND nodesync = {'enabled': 'true', 'incremental': 'true'}
AND read_repair = 'BLOCKING'
AND speculative_retry = '99PERCENTILE';
```

A new key has been created in the KMIP vault:

```
UUID: e9c264bd-c145-4ec1-a88a-67cf2c991631
Object Type: Symmetric Key
State: ACTIVE
Activation Date: Sep 7, 2024, 12:06:18 AM
Cryptographic Usage Mask: Encrypt,Decrypt
Key Format Type: Raw
Cryptographic Algorithm: AES
Cryptographic Length: 128
Encrypted with KEK: No
Initial Date: Sep 7, 2024, 12:06:18 AM Last Status
Changed Date: Sep 7, 2024, 12:06:18 AM x-dse-version: 6.9.2
```

7. Display table content.

```
cqlsh> SELECT * FROM my_keyspace.customers;

customer_id                          | email                | first_name    | last_name
-------------------------------------+----------------------+---------------+-----------
2056af70-0f0a-4d4d-a8a6-561883c523a0    | jane.doe@example.com  | Jane          | Doe
```

```
dfcb3df4-4f26-4df9-98aa-a035f8149641    | john.doe@example.com  | John          | Doe
```

Chapter 3. Additional resources and related products