EDB Postgres and Entrust KeyControl

Integration Guide

2024-03-05

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 (HSM) to securely store keys.

Using EDB Postgres Advanced Server or EDB Postgres Extended Server TDE capabilities along with Entrust KeyControl for key management protect sensitive data wherever those data reside.

1.1. Documents to read first

- Entrust KeyControl Online Documentation Set
- Installing Postgres
- PostgreSQL Linux downloads (Red Hat family)
- Installing PostgreSQL on AlmaLinux9 or Rocky Linux 9 x86_64 EDB Postgres Advanced Server (EPAS)
- Transparent Data Encryption overview
- Transparent Data Encryption
- PyKMIP Installation
- Uninstalling EDB Postgres Advanced Server on Linux v16
- How to Set Up a Firewall Using firewalld on Rocky Linux 9

1.2. Product configuration

Product	Version	Notes
EDB Postgres	15 or later	
KeyControl KMS	10.1.1	KMIP Vault installed and deployed per your environment
Python	3.9 or later	Installed on the Linux server
Netcat		Installed on the Linux server for troubleshooting

Chapter 2. Procedures

High-level overview of implementing Postgres TDE using KeyControl:

1. Install Postgres on AlmaLinux 9 or Rocky Linux 9 x86_64
2. Install PyKMIP on the Postgres Linux server
3. Create a CSR for the Postgres Linux server
4. Generate a KMIP client certificate for the Postgres Linux server
5. Configure PyKMIP on the Postgres Linux server
6. Test the KMIP service
7. Install edb-tde-kmip-client and check prerequistes
8. Perform the initial configuration and create an encrypted Postgres database
9. Check the database access

2.1. Install Postgres on AlmaLinux 9 or Rocky Linux 9 x86_64

1. Determine if your repository exists:

   ```
   $ dnf repolist | grep enterprisedb
   ```

   If no output is generated, the repository isn't installed.

2. Go to EDB repositories (https://www.enterprisedb.com/repos-downloads).
3. Select the button that provides access to the EDB repository.
4. Select the platform and software that you want to download.
5. Follow the instructions for setting up the EDB repository.
6. Select **EDB Postgres Advanced Server**.
7. Install EDB Repository

   ```
   [enterprisedb@localhost ~]$ curl -1sLf
   'https://downloads.enterprisedb.com/TAu5BEpU2oKFuRuaRUsdRymZZ9mROuUQ/enterprise/setup.
   rpm.sh' | sudo -E bash
   [sudo] password for enterprisedb:
   Executing the setup script for the 'enterprisedb/enterprise' repository ...
   OK: Checking for required executable 'curl' ...
   OK: Checking for required executable 'rpm' ...
   OK: Detecting your OS distribution and release using system methods ...
   ^^^^: ... Detected/provided for your OS/distribution, version and architecture:
   > > > >:
   > > > >: ... distro=rocky version=9.0 codename=Blue arch=x86_64
   > > > >:
   > > > > OK: Importing 'enterprisedb/enterprise' repository GPG keys into rpm ...
   ```

```
> > > > OK: Checking for available package manager (DNF/Microdnf/YUM/Zypper) ...
> > > > ^^^^: ... Detected package manager as 'dnf'
> > > > OK: Checking for dnf dependency 'dnf-plugins-core' ...
> > > > OK: Checking if upstream install config is OK ...
> > > > OK: Fetching 'enterprisedb/enterprise' repository configuration ...
> > > > OK: Installing 'enterprisedb/enterprise' repository via dnf ...
> > > > RUN: Updating the dnf cache to fetch the new repository metadata ...Importing GPG key 0x9F1EF813:
> > > > enterprisedbid : "Cloudsmith Package (enterprisedb/enterprise) <support@cloudsmith.io>"
> > > > Fingerprint: 31A4 CF09 0B3A E265 F131 58DE E71E B082 9F1E F813
> > > > From : https://downloads.enterprisedb.com/TAu5BEpU2oKFuRuaRUsdRymZZ9mROuUQ/enterprise/gpg.E71
EB0829F1EF813.key
> > > > Importing GPG key 0x9F1EF813:
> > > > enterprisedbid : "Cloudsmith Package (enterprisedb/enterprise) <support@cloudsmith.io>"
> > > > Fingerprint: 31A4 CF09 0B3A E265 F131 58DE E71E B082 9F1E F813
> > > > From : https://downloads.enterprisedb.com/TAu5BEpU2oKFuRuaRUsdRymZZ9mROuUQ/enterprise/gpg.E71
EB0829F1EF813.key
> > > > Importing GPG key 0x9F1EF813:
> > > > enterprisedbid : "Cloudsmith Package (enterprisedb/enterprise) <support@cloudsmith.io>"
> > > > Fingerprint: 31A4 CF09 0B3A E265 F131 58DE E71E B082 9F1E F813
> > > > From : https://downloads.enterprisedb.com/TAu5BEpU2oKFuRuaRUsdRymZZ9mROuUQ/enterprise/gpg.E71
EB0829F1EF813.key
> > > > OK: Updating the dnf cache to fetch the new repository metadata ...

OK: The repository has been installed successfully -You're ready to rock!
```

8. Determine if your repository exists:

```
[enterprisedb@localhost ~]$ dnf repolist | grep enterprisedb
enterprisedb-enterprise enterprisedb-enterprise
enterprisedb-enterprise-noarch enterprisedb-enterprise-noarch
enterprisedb-enterprise-source enterprisedb-enterprise-source
```

9. Install the EPEL repository:

```
$ sudo dnf -y install epel-release
```

10. Enable additional repositories to resolve dependencies:

```
$ sudo dnf config-manager --enable crb --> OK
```

11. Enable additional repositories to resolve dependencies:

```
$ dnf config-manager --set-enabled powertools
```

12. Disable the built-in Postgres module:

```
$ dnf -qy module disable postgresql
```

13. Install the postgres edb server package

```
$ sudo dnf -y install edb-as16-server
```

2.2. Install PyKMIP on the Postgres Linux server

PyKMIP must be installed as `enterprisedb`.

1. Check or install Python on the server:

```
[enterprisedb@localhost ~]$ python
Python 3.9.10 (main, Feb 9 2022, 00:00:00)
[GCC 11.2.1 20220127 (Red Hat 11.2.1-9)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

2. Install `pip`:

```
[enterprisedb@localhost ~]$ sudo dnf install pip
```

3. Upgrade `pip`:

```
[enterprisedb@localhost ~]$ /usr/bin/python3.9 -m pip install --upgrade pip
```

4. Upgrade `setuptools`:

```
[enterprisedb@localhost ~]$ sudo pip3 install --upgrade setuptools
```

5. Install `tox`:

```
[enterprisedb@localhost ~]$ pip install tox
```

6. Install `git`:

```
[enterprisedb@localhost ~]$ sudo dnf install -y git
```

7. Install `cryptography`:

```
[enterprisedb@localhost ~]$ sudo pip install cryptography
```

8. Build PyKMIP:

```
[enterprisedb@localhost ~]$ git clone https://github.com/openkmip/pykmip.git
[enterprisedb@localhost ~]$ sudo python3.9 pykmip/setup.py install
```

9. Run the PyKMIP tests:

```
[enterprisedb@localhost pykmip]$ bin/run_tests.sh
```

2.3. Create a CSR for the Postgres Linux server

1. Change your working directory to the `$home` directory of `enterprisedb`:

```
[enterprisedb@client1 ~]$ cd ~
```

2. Create a new certificate request good for 3 years:

```
[enterprisedb@client1 ~]$ openssl req -new -nodes -keyout client1.key -out client1.csr
-days 1095
Ignoring -days; not generating a certificate
Generating a RSA private key
.....................................................................................
+++++
.....................................................................................
+++++
writing new private key to 'client1.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]:FR
State or Province Name (full name) []:Paris
Locality Name (eg, city) [Default City]:Paris
Organization Name (eg, company) [Default Company Ltd]:Entrust
Organizational Unit Name (eg, section) []:DPS
Common Name (eg, your name or your server's hostname) []:client1.ncipher.com
Email Address []:guillaume.cesbron@entrust.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:<empty>
An optional company name []:
```

3. Verify that the CSR has been generated.

```
[enterprisedb@kmip CA]$ ls client1.csr
```

4. Download the CSR file on the local machine. You will use it to generate the KMIP client certificate.

2.4. Generate a KMIP client certificate for the Postgres Linux server

1. Sign in to the KeyControl KMIP vault webGUI using an account with Security Admin privileges.

```
https://<keycontrol-vault-ip-address>/kmip/#/login
```

2. Navigate to **Security > Client certificates**.
3. Select the **+** button.
4. In **Create Client Certificate**, specify the options you want to use:

Certificate Name	`client1`
Certificate Expiration	`3 years from today`
Certificate Signing Request (CSR)	`client1.csr`
Encrypt Certificate Bundle	`No`

5. Select **Create**.

   A message is displayed that the certificate has been successfully generated.

6. Select the new certificate from the options.
7. Select **Download Certificate**.

   A new file is downloaded by the browser.

   The `.zip` file contains a `enterprisedb` certification/key file called `enterprisedbname.pem` and a server certification file called `cacert.pem`.

8. Unzip the bundled file:

```
$ unzip client1_2023-12-15-16-54-42.zip
Archive: client1_2023-12-15-16-54-42.zip
inflating: client1.pem
inflating: cacert.pem
```

9. Upload the 2 pem files to the PostgreSQL Linux machine:

```
$ scp client1.pem enterprisedb@<postgres-server-ip-address>:.
$ scp cacert.pem enterprisedb@<postgres-server-ip-address>:.
```

2.5. Configure PyKMIP on the Postgres Linux server

PyKMIP must be configured to use the client certificate generated by KeyControl.

1. Sign in to the Linux server:

```
$ ssh enterprisedb@<postgres-server-ip-address>
```

2. Install KMIP client and CA certificates:

```
[enterprisedb@<postgres-server-ip-address>~]$ sudo mkdir -p /etc/pykmip/certs
[enterprisedb@<postgres-server-ip-address>~]$ sudo cp client1.pem /etc/pykmip/certs/client_cert.pem
[enterprisedb@<postgres-server-ip-address>~]$ sudo cp client1.key/etc/pykmip/certs/client_private_key.pem
[enterprisedb@<postgres-server-ip-address>~]$ sudo cp cacert.pem /etc/pykmip/certs/server_ca_cert.pem
```

3. Create the `PYTHONPATH` variable:

```
[enterprisedb@<postgres-server-ip-address>~]$ vim ~/.bash_profile
#PYTHONPATH="$PYTHONPATH:/home/enterprisedb/pykmip"
PYTHONPATH="/home/enterprisedb/pykmip"
export PYTHONPATH
[enterprisedb@<postgres-server-ip-address> ~]$ source ~/.bash_profile
```

4. You need now to edit the `pykmip client.conf` file that must be placed in `/etc/pykmip`.

   The `policy.jsonfile` must be also placed there.

   Create the file `/etc/pykmip/pykmip.conf`:

   

   Remember to comment out the last two lines.

```
[enterprisedb@<postgres-server-ip-address>~]$ sudo chmod a+r /etc/pykmip/certs/*
[enterprisedb@<postgres-server-ip-address> ~]$ sudo cp ~/pykmip/examples/pykmip.conf
/etc/pykmip/pykmip.conf
[enterprisedb@<postgres-server-ip-address> ~]$ sudo vim /etc/pykmip/pykmip.conf
[client]
host=<keycontrol-vault-ip-address>
port=5696
certfile=/home/enterprisedb/pykmip/kmip.pem
keyfile=/home/enterprisedb/pykmip/kmip.key
ca_certs=/home/enterprisedb/pykmip/server_ca_certs.pem
cert_reqs=CERT_REQUIRED
ssl_version=PROTOCOL_SSLv23
do_handshake_on_connect=True
suppress_ragged_eofs=True
#enterprisedbname=example_enterprisedbname
#password=example_password
```

2.6. Test the KMIP service

1. Ping the KeyControl appliance:

```
$ ping <keycontrol-vault-ip-address>
PING <keycontrol-vault-ip-address> (192.168.1.235) 56(84) bytes of data.
64 bytes from <keycontrol-vault-ip-address> (192.168.1.235): icmp_seq=1 ttl=64 time=0.701 ms
64 bytes from <keycontrol-vault-ip-address> (192.168.1.235): icmp_seq=2 ttl=64 time=0.437 ms
64 bytes from <keycontrol-vault-ip-address> (192.168.1.235): icmp_seq=3 ttl=64 time=0.421 ms
64 bytes from <keycontrol-vault-ip-address> (192.168.1.235): icmp_seq=4 ttl=64 time=0.405 ms
^C
```

```
--- <keycontrol-vault-ip-address> ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3044ms
```

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

```
[enterprisedb@esx-lnx-srv-02 pykmip]$ echo | openssl s_client -servername <keycontrol-vault-ip-address>
-connect <keycontrol-vault-ip-address>:443 2>/dev/null | openssl x509 -noout -issuer -subject -dates -
ext subjectAltName -fingerprint
issuer=DC = org, DC = simple, O = Simple Inc, OU = Simple Signing CA, CN = Simple
Signing CA
subject=CN = <keycontrol-vault-ip-address>
notBefore=Sep 1 10:29:46 2022 GMT
notAfter=Aug 31 10:29:46 2024 GMT
X509v3 Subject Alternative Name:
DNS:hytrust-kc-prod.mylab.local
SHA1 Fingerprint=24:5C:DC:B5:07:29:E1:7D:9D:89:6D:DF:47:AC:E1:3B:48:CC:13:0D
```

3. Test the CA certificate:

```
$ openssl s_client -CAfile /home/enterprisedb/pykmip/server_ca_certs.pem -connect
<keycontrol-vault-ip-address>: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:
20606FADDD52929948B0D754D19F412FA032ACB4859D7E5B0B689BFCCB2BE73D5566EA58F8EA5837E16821
0046FA5AE4
PSK identity: None
PSK identity hint: None
SRP enterprisedbname: 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
```

4. Create an AES 256 AES key:

```
[enterprisedb@esx-lnx-srv-02 pykmip]$ cd ~/pykmip/
enterprisedb@esx-lnx-srv-02 pykmip]$ python3.9 pykmip/kmip/demos/pie/create.py -a AES -l 256
2023-12-15 18:23:38,167 - demo - INFO - Successfully created symmetric key with ID: b36ad09f-d27c-4052-
88ae-8952caa55467
```

5. Create and activate an AES 26 key as the user `enterprisedb`:

```
[enterprisedb@esx-lnx-srv-02 pykmip]$ python3.9
pykmip/kmip/demos/pie/create_and_activate.py -a AES -l 256
2023-12-17 19:38:12,029 - demo - INFO - Successfully created symmetric key with ID:
bea90149-3426-4a6f-82a5-bb24e4cbe4d3
2023-12-17 19:38:12,054 - demo - INFO - Successfully activated symmetric key with ID:
bea90149-3426-4a6f-82a5-bb24e4cbe4d3
```

   The key must be activated otherwise it cannot be used for encryption.

2.7. Install edb-tde-kmip-client and check prerequistes

1. Install `edb-tde-kmip-client` on your system:

```
$ sudo dnf install -y edb-tde-kmip-client
```

2. Locate `edb_tde_kmip_client.py` script:

```
[root@localhost ~]# find / -name edb_tde_kmip_client.py
/usr/edb/kmip/client/edb_tde_kmip_client.py
```

3. Verify encryption with an active key using the `pykmip` variant:

```
[root@localhost ~]# printf secret | python3.9
/usr/edb/kmip/client/edb_tde_kmip_client.py encrypt --out-file=test.bin
  --pykmipconfig-file=/etc/pykmip/pykmip.conf
  --key-uid='bea90149-3426-4a6f-82a5-bb24e4cbe4d3' --variant=pykmip
```

Location of the KMIP client	`/usr/edb/kmip/client/edb_tde_kmip_client.py encrypt`
Output file	`test.bin`
Location of PyKMIP configuration file	`/etc/pykmip/pykmip.conf`
Encrypted key output	TDE key output

4. Verify decryption with the same key:

```
[root@localhost ~]# python3.9 /usr/edb/kmip/client/edb_tde_kmip_client.py decrypt --in-file=test.bin
--pykmip-config-file=/etc/pykmip/pykmip.conf --key-uid='bea90149-3426-4a6f-82a5-bb24e4cbe4d3'
--variant=pykmip
secret
```

2.8. Perform the initial configuration and create an encrypted Postgres database

After you create the key and verify encryption and decryption, you can export `PGDATAKEYWRAPCMD` and `PGDATAKEYUNWRAPCMD` to wrap and unwrap your encryption key and initialize your database.

1. Sign in to your EDB Postgres distribution system as the database superuser: In this example, it's the `enterprisedb` user:

```
$ sudo su - enterprisedb
```

2. Navigate to the `/bin` directory of your executables:

   In this example, it's `/usr/lib/edbas/16/bin`.

```
$ cd /usr/edb/as16/bin
```

3. Perform the initial configuration of the database:

```
$ export PGDATAKEYWRAPCMD='python3.9 /usr/edb/kmip/client/edb_tde_kmip_client.py
encrypt --out-file=test.bin --pykmip-config-file=/etc/pykmip/pykmip.conf --keyuid='
bea90149-3426-4a6f-82a5-bb24e4cbe4d3' --variant=pykmip'
$ export PGDATAKEYUNWRAPCMD='python3.9 /usr/edb/kmip/client/edb_tde_kmip_client.py
decrypt --pykmip-config-file=/etc/pykmip/pykmip.conf --key-uid=bea90149-3426-4a6f-
82a5-bb24e4cbe4d3 --in-file=%p --variant=pykmip'
$ ./initdb -D /var/lib/edb/as16/data -y
```

2.9. Check the database access

1. Create a database named hr to hold human resource information in psql:

```
edb=# CREATE DATABASE hr;
CREATE DATABASE
```

2. Connect to the hr database inside psql:

```
edb=# \c hr
psql (16.1.0, server 16.1.0)
You are now connected to database "hr" as enterprisedb "enterprisedb"
```

3. Create columns to hold department numbers, unique department names, and locations:

```
hr=# CREATE TABLE public.dept (deptno numeric(2) NOT NULL CONSTRAINT dept_pk
PRIMARY KEY, dname varchar(14) CONSTRAINT dept_dname_uq UNIQUE, loc
varchar(13));
CREATE TABLE
```

4. Insert values into the dept table:- OK

```
hr=# INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
INSERT 0 1
hr=# INSERT into dept VALUES (20,'RESEARCH','DALLAS');
INSERT 0 1
```

5. View the table data by selecting the values from the table:

```
hr=# SELECT * FROM dept;
deptno | dname | loc
10 | ACCOUNTING | NEW YORK
20 | RESEARCH | DALLAS
(2 rows)
```

Chapter 3. Additional resources and related products

3.1. KeyControl

3.2. Entrust digital security solutions

3.3. nShield product documentation