



Thales CipherTrust Transparent Encryption Implementation Guide

Release 1.0.0

EDB

Jun 24, 2022

CONTENTS

1	‘Partner Information’	3
2	‘Solution Summary’	5
3	‘Configuring the Integration’	7
3.1	Prerequisites	7
3.2	Postgres Host	7
3.3	CipherTrust Manager	7
3.4	Configuring CipherTrust Manager Logon to the CipherTrust Manager (CM) Web GUI and perform the following steps:	7
3.5	Installing CTE Agent	8
4	‘Using Thales CipherTrust Transparent Encryption’	9
4.1	Sample User Scenarios	9
5	‘Certification Environment’	11
6	‘Appendix’	13
6.1	Sample config.yml file	13

EDB GlobalConnect Technology Partner Implementation Guide

Thales CipherTrust Transparent Encryption

‘PARTNER INFORMATION’

1	2
Partner Name	Thales
Partner Product	CipherTrust Transparent Encryption
Web Site	https://cpl.thalesgroup.com/encryption
Version & Platform	7.1.0, Available platforms: Windows, Linux
Product Description	CipherTrust Transparent Encryption (CTE) delivers data-at-rest encryption with centralized key management, privileged user access control, and detailed data access audit logging. This protects data wherever it resides: on-premises, across multiple clouds and within big data, and container environments.

‘SOLUTION SUMMARY’

Thales’ CipherTrust Transparent Encryption secures data at-rest for Postgres databases and backups with file system-level encryption backed by centralized key management, privileged user access controls, and detailed data access audit logging. CipherTrust Transparent Encryption allows customers to adopt Postgres for highly-sensitive and regulated data both on-premises and in the cloud while also meeting their compliance obligations. CipherTrust Transparent Encryption has been certified with EDB Postgres Advanced Server, and with EDB Postgres Extended Server as part of a BDR (bi-directional replication) cluster, and with Barman.

Note: EDB Postgres Extended Server represents EDB Postgres Extended Server* with BDR (Bi-Directional Replication) and Barman.

‘CONFIGURING THE INTEGRATION’

Implementing the CipherTrust Transparent Encryption (CTE) solution requires the following components:

- Postgres server installed and operational.
- CipherTrust Manager installed and operational.
- A CTE agent installed on the Postgres host registered to the CipherTrust Manager.

The following diagram shows the basic flow of the CTE solution:

3.1 Prerequisites

3.2 Postgres Host

- Ensure that the Postgres server is installed and running.
- For CentOS 7, you need to install the following repository:

```
sudo yum install -y lsof
```

3.3 CipherTrust Manager

1. Ensure CipherTrust Manager is installed and running.

3.4 Configuring CipherTrust Manager Logon to the CipherTrust Manager (CM) Web GUI and perform the following steps:

1. Create a registration token.
 - a. Navigate to **Key and Access Management** and select **Registration Tokens**. This token is used for the CTE agent enrollment to CM.
 - b. Select **New Registration Token** to create a new registration token.

The following screenshot shows a registration token created with the name **edb**.

2. Create user sets.
 - a. Navigate to CTE and select Policies, Policy Elements and then User Sets.
 - b. Select Create User Set to create a new user set.

Create the Postgres, EnterpriseDB and Barman user sets as shown in the following screenshots.

3. CreatePolicies

- a. Navigate back to **Policies** and select **CreatePolicy**.

The following screenshot shows Live Data Transformation (LDT) policies postgres-policy, epas-policy and barman-policy.

Note: The policies include the User Sets **Postgres** and **EnterpriseDB** respectively created in Step 2 and the same Key Rule for the policies:

3.5 Installing CTE Agent

Refer to the following guides from Thales for installing the CTE agent on the Postgres host:

[CTE Agent Quick Start Guide](#)

[CTE Agent Advanced Installation Guide](#)

Note: You will need the Registration Token and host address of the CipherTrust Manager during the installation.

After the CTE agent is successfully installed, verify the Postgres host is registered with CM.

1. Log on to the CM Web GUI and navigate to **CTE**.
2. Select **Clients**. The client status should appear as **Healthy** as shown below (you may have to wait a few seconds for the status to get updated).

The following screenshot shows clients registered with the CipherTrust Manager.

‘USING THALES CIPHERTRUST TRANSPARENT ENCRYPTION’

CTE protects data either at the file level or at the storage device level. A CTE Agent running on the (Postgres) host manages the files behind a GuardPoint by enforcing the policy associated with it, and communicates data access events to the CipherTrust Manager for logging. A GuardPoint is usually associated with a Linux mount point or a Windows volume, but may also be associated with a directory subtree.

The following diagram shows the CTE architecture.

4.1 Sample User Scenarios

This section describes sample user scenarios of deploying CTE solutions on EDB Postgres Advanced Server and EDB Postgres Extended Server with BDR hosts.

- **EDBPostgresAdvancedServer**
- **EDBPostgresExtendedServerwithBDR**

EDBPostgresAdvancedServer(SingleInstance)

1. Install CTE agent on the Postgres host.
2. Login to the Postgres host and stop the postgres server.
3. Create the GuardPoints via the CM Web GUI using the **epas-policy** Policy on the postgres host. Set the following directories as the **ProtectedPath** on the EDB Postgres Advanced Server host (assuming PGDATA is set /var/lib/edb/as13/data on the host):
4. Restart the Postgres server on the Postgres host as the user **enterprisedb** . Make sure you are logged in using ssh (not sudo).

EDBPostgresExtendedServerwithBDR-Always-ON

The following diagram shows the BDR-Always-ON architecture. For more details, refer to the [BDR-Always-ON Architecture](#) documentation.

Note: The documentation requires EDB access credentials.

1. Install CTE agents on all the postgres and barman nodes.
2. Create a GuardPoint via the CM Web GUI using the **barman-policy** Policy on the directory /var/lib/barman/<server-name> on the barman node in data center A (DC A). The following screenshot shows a GuardPoint created for the barman node.
3. Login to the Standby node in data center A and stop the postgres server.

4. Create a GuardPoint on the Standby node via the CM Web GUI using the postgres-policy Policy on the PGDATA directory `/opt/postgres/data`.
5. Restart the Postgres server on the Standby node as the user **postgres**. Make sure you are logged in using ssh (not sudo).
6. Login to the Shadow Master node in data center A and stop the postgres server.
7. Create a GuardPoint on the Shadow Master node via the CM Web GUI using the postgres-policy Policy on the PGDATA directory `/opt/postgres/data`.
8. Restart the Postgres server on the Shadow Master node as the user **postgres**. Make sure you are logged in using ssh (not sudo).
9. Login to the Lead Master node in data center A and stop the postgres server.
10. Create a GuardPoint on the Lead Master node via the CM Web GUI using the postgres-policy Policy on the PGDATA directory `/opt/postgres/data`.
11. Restart the Postgres server on the Lead Master node as the user **postgres**. Make sure you are logged in using ssh (not sudo).

The following screenshot shows a GuardPoint created for Lead Master in data center A.

12. Repeat steps 2 through 11 for postgres and barman nodes in data center B (DC B).

‘CERTIFICATION ENVIRONMENT’

1	2
Certification Test Date	May 19 2021
EDB Postgres Advanced Server	13.2.5
OS	CentOS Linux 7 (Core)
Memory	2G
Processor	Intel® Xeon® Processor SP Family (“Skylake”)
Cloud Platform	OpenStack (Kilo)
CPU(s)	1
Core(s) per socket	1
Socket(s)	1
Storage	80 GB
CipherTrust Transparent Encryption	7.0.0.99

1	2
Certification Test Date	May 19 2021
EDB Postgres Extended Server	11
Postgres-BDR	3.6.25
OS	CentOS Linux 7
Cloud Platform	AWS
Deployment Tool	tpaexec v20.11

Note: Refer to the sample config.yml file in the Appendix for deployment details.

‘APPENDIX’

6.1 Sample config.yml file

```
- --
architecture: BDR-Always-ON
cluster_name: bdrtest-always-on-aws
cluster_tags:
  Owner: Aziz Rahman
cluster_bucket: bdr-alwayson-azizrahman

cluster_rules:

-   cidr_ip: 0.0.0.0/0
    from_port: 22
    proto: tcp
    to_port: 22

-   cidr_ip: 10.35.104.176/24
    from_port: 0
    proto: tcp
    to_port: 65535

-   cidr_ip: 10.35.250.208/24
    from_port: 0
    proto: tcp
    to_port: 65535
ec2_ami:
  Name: RHEL-7.8_HVM_GA-20200225-x86_64-1-Hourly2-GP2
  Owner: 309956199498
ec2_instance_reachability: public
ec2_vpc:
  Name: BDR-Test
  cidr: 10.35.0.0/16

cluster_vars:
  bdr_database: bdrdb
  bdr_node_group: bdrgroup
  enable_harp: false
  extra_postgres_extensions:
    - pglogical
```

(continues on next page)

(continued from previous page)

```

postgres_coredump_filter: 0xff
preferred_python_version: python3
repmgr_failover: manual
tpa_2q_repositories:
- products/2ndqpostgres/release
packages:
  RedHat:
    - lsof

use_volatile_subscriptions: false

locations:

- Name: a
  az: us-east-1a
  region: us-east-1
  subnet: 10.35.104.0/24

- Name: b
  az: us-east-1b
  region: us-east-1
  subnet: 10.35.250.0/24

instance_defaults:
  default_volumes:
    - device_name: root
      encrypted: true
      volume_size: 16
      volume_type: gp2
    - device_name: /dev/xvdf
      encrypted: true
      vars:
        volume_for: postgres_data
        volume_size: 64
        volume_type: gp2
  platform: aws
  type: t3.micro
  vars:
    ansible_user: ec2-user

instances:

- Name: bdr1-a
  backup: barman-a
  location: a
  node: 1
  role:
    - primary
    - bdr

- Name: bdr2-a
  location: a

```

(continues on next page)

(continued from previous page)

```
node: 2
role:
- primary
- bdr

- Name: bdr3-b
  backup: barman-b
  location: b
  node: 3
  role:
  - primary
  - bdr

- Name: bdr4-b
  location: b
  node: 4
  role:
  - primary
  - bdr

- Name: standby-a
  location: a
  node: 5
  role:
  - primary
  - bdr
  - readonly
  upstream: bdr2-a

- Name: standby-b
  location: b
  node: 6
  role:
  - primary
  - bdr
  - readonly
  upstream: bdr4-b

- Name: barman-a
  location: a
  node: 7
  role:
  - barman
  volumes:
  - device_name: /dev/xvdf
    encrypted: true
    vars:
      volume_for: barman_data
      volume_size: 128
      volume_type: gp2

- Name: barman-b
```

(continues on next page)

(continued from previous page)

```
location: b
node: 8
role:
- barman
volumes:
- device_name: /dev/xvdf
  encrypted: true
  vars:
    volume_for: barman_data
  volume_size: 128
  volume_type: gp2

- Name: pgbouncer-haproxy-a
location: a
node: 9
role:
- pgbouncer
- haproxy
vars:
  haproxy_backend_servers:
    - bdr1-a
    - bdr2-a
volumes:
- device_name: /dev/xvdf
  volume_type: none

- Name: pgbouncer-haproxy-b
location: b
node: 10
role:
- pgbouncer
- haproxy
vars:
  haproxy_backend_servers:
    - bdr3-b
    - bdr4-b
volumes:
- device_name: /dev/xvdf
  volume_type: none
```