2015 年 6 月 15 日

Transparent Data Encryption for PostgreSQL

Free Edition 検証結果

日本ヒューレット・パッカード株式会社
篠田典良

目次

1. 本文書について

1.1 本文書の概要

本文書は、2015 年 6 月 5 日に日本電気株式会社（NEC）様が公開した「Transparent Data Encryption for PostgreSQL」について、独自に検証を行った結果をまとめた資料です。

「Transparent Data Encryption for PostgreSQL（以下 TDE）」は PostgreSQL データベース内に保存されたデータに対して自動的に暗号化／復号を行う機能を提供します。本文書では、TDE のインストール、初期設定、実行方法を示し、Oracle Database が持つ同等機能との比較を行っています。

1.2 本文書の対応バージョン

本文書で検証を行った環境は以下のバージョンを使用しています。オンライン・マニュアルに示された PostgreSQL のバージョンが PostgreSQL 9.3 であったため、最新版の PostgreSQL 9.4 ではテストしていません。

表 1 対象ソフトウェアとバージョン

種別	バージョン
データベース製品	PostgreSQL 9.3.8
オペレーティング・システム	Red Hat Enterprise Linux 6 Update 5 (x86-64)
TDE ソフトウェア	Transparent Data Encryption for PostgreSQL Free Edition

1.3 文責

本文書の内容は、日本ヒューレット・パッカード株式会社の公式見解ではありません。著者および所属会社は、本文書の内容の間違いにより生じた問題に対して責任を負いません。本文書に対するご意見／ご要望は、日本ヒューレット・パッカード株式会社 コンサルティング事業統括 篠田典良（noriyoshi.shinoda@hp.com）までご連絡ください。

2. TDE 検証結果

本章では TDE の検証結果を記載しています。

2.1 TDE 概要

2.1.1 TDE とは

Transparent Data Encryption for PostgreSQL (TDE) は PostgreSQL に格納されたデータに対して自動的に暗号化／復号を行う機能を提供します。PostgreSQL には標準で暗号化に関する機能として pgcrypto Contrib モジュールが提供されています。しかし pgcrypto は暗号化／復号を行う関数を提供しているだけであるため、データベース利用者が独自に関数を実行して暗号化／復号を行う必要があります。これに対して TDE は利用者がデータの暗号化を意識せず、従来通りの SQL 文を実行するだけで復号されたデータに自動的にアクセスできます。

□ エディション

TDE には以下のエディションが提供されています。

- Transparent Data Encryption for PostgreSQL Free Edition
  オープンソースで公開された TDE モジュール本体です。
  ライセンスは GPL3 を採用しています。
- Transparent Data Encryption for PostgreSQL Enterprise Edition
  Free Edition に加えて、対象データ型の追加、データベース診断機能、データベース復旧機能、暗号鍵管理機能等を搭載した有償エディションです。

□ 参考情報

本文書の作成には以下の URL を参考にしました。

- プレスリリース
  http://jpn.nec.com/press/201506/20150605_01.html
- GitHub
  https://github.com/nec-postgres/tdeforpg/
- オンライン・マニュアル
  https://github.com/nec-postgres/tdeforpg/wiki/Manual(JP)

2.1.2 TDE が対応するリスク

PostgreSQL に対する接続ユーザー名／パスワードの漏洩からデータの参照や改ざんを防ぐことができます。TDE を利用するためには、PostgreSQL インスタンスに接続後、暗号化鍵をオープンする関数に暗号鍵情報を入力して実行する必要があります。暗号鍵情報を指定していないセッションは、たとえ PostgreSQL インスタンスに接続できたとしても、暗号化された列データを参照できません。

またバックアップしたオフライン・データが漏洩された場合でも暗号鍵が解析されない限り情報にアクセスすることができません。

2.2 インストール

オンライン・マニュアル内に TDE のインストール手順が示されています。

2.2.1 準備

PostgreSQL 9.3 と TDE のソースコードを準備し、環境変数を設定します。

表 2 環境変数

環境変数	バージョン
TDEHOME	TDE ソースコードのディレクトリ
PGSRC	PostgreSQL ソースコードのディレクトリ
PGHOME	TDE をインストールする PostgreSQL ディレクトリ

例 1 環境変数の設定

```
$ export TDEHOME=${HOME}/tdeforpg-master/
$ export PGSRC=${HOME}/postgresql-9.3.8/
$ export PGHOME=${HOME}/pg938/
$ cd $PGSRC
$ ./configure
<<以下省略>>
```

TDE のソースコードをビルドし、ライブラリを作成します。

例 2 ビルドとライブラリの設定

```
# ln -s $PGHOME/lib/pgcrypto.so /usr/lib64/libpgcrypto.so
#
$ cd $TDEHOME/SOURCES/data_encryption
$ sh makedencryption.sh 93 $PGSRC
rm -f data_encryption.so data_encryption.o
/usr/bin/gcc -c -Wall -O3 -fPIC -I/home/postgres/postgresql-9.3.8//src/include
-I/home/postgres/postgresql-9.3.8//contrib/pgcrypto data_encryption.c
/usr/bin/gcc    -shared    -o    data_encryption.so    data_encryption.o    -
L/home/postgres/postgresql-9.3.8//contrib/pgcrypto      -lpgcrypto      -Wl,-
rpath,'/home/postgres/postgresql-9.3.8//contrib/pgcrypto'
        linux-vdso.so.1 =>  (0x00007fff58dff000)
        libpgcrypto.so => /usr/lib64/libpgcrypto.so (0x00007f18c26ee000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f18c2359000)
        libz.so.1 => /lib64/libz.so.1 (0x00007f18c2143000)
        /lib64/ld-linux-x86-64.so.2 (0x0000003aed800000)

INFO: data_encryption.so was made.
$
# ln -s tdeforpg-master/SOURCES/data_encryption/93/data_encryption93.so
        /usr/lib64/data_encryption.so
```

PostgreSQL のパラメーターshared_preload_libraries に、ビルドされたライブラリを登録し、インスタンスを再起動します。

例 3 ライブラリの登録

```
$ grep shared_preload_libraries data/postgresql.conf
shared_preload_libraries = '/usr/lib64/data_encryption.so'
$ pg_ctl restart -D data -w -m fast
waiting for server to shut down.... done
server stopped
waiting for server to start....
LOG:  loaded library "/usr/lib64/data_encryption.so"
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
 done
server started
```

TDE を使用する PostgreSQL データベースに pgcrypto モジュールを登録します。

例 4 EXTENSION の登録

```
$ psql -d postgres -U postgres
postgres=# CREATE EXTENSION pgcrypto ;
CREATE EXTENSION
```

CREATE EXTENSION pgcrypto が実行されていないデータベースに対して暗号鍵を設定しようとすると、以下のエラーが発生します。

例 5 TDE 導入エラー

```
ERROR:  function pgp_sym_encrypt(text, text, unknown) does not exist
LINE 1: INSERT INTO cipher_key_table VALUES(pgp_sym_encrypt(cipher_k...
                                            ^

HINT:  No function matches the given name and argument types. You might need to add explicit type casts.
QUERY:  INSERT INTO cipher_key_table VALUES(pgp_sym_encrypt(cipher_key, cipher_key, 'cipher-algo=aes256, s2k-mode=1'), cipher_algorithm)
CONTEXT:  PL/pgSQL function cipher_key_regist(text,text,text) line 59 at SQL statement
```

2.2.2 TDE の導入

TDE を対象データベースに導入します。導入には cipher_setup.sh スクリプトを PostgreSQL 管理 OS ユーザーの権限で実行します。以下の項目に対する入力が必要です。

表 3　入力項目

項目	プロンプト	備考
メニュー番号	Transparent data encryption feature setup menu	
ポート番号	Please enter database server port to connect	
ユーザー名	Please enter database user name to connect	管理者権限
パスワード	Please enter password for authentication	非表示
データベース名	Please enter database name to connect	

例 6 TDE 導入

```
$ cd $TDEHOME/SOURCES
$ sh bin/cipher_setup.sh $PGHOME
Transparent data encryption feature setup script
Please select from the setup menu below
Transparent data encryption feature setup menu
1: activate  the transparent data encryption feature
2: inactivate the transparent data encryption feature
select menu [1 - 2] > 1
Please enter database server port to connect : 5432
Please enter database user name to connect : postgres
Please enter password for authentication : <<非表示>>
Please enter database name to connect : postgres
CREATE LANGUAGE
INFO: Transparent data encryption feature has been activated
```

2.2.3 暗号鍵の設定

TDE の導入が成功したら、暗号鍵の生成を行います。暗号鍵の生成には cipher_key_regist.sh スクリプトを実行し、暗号鍵となる文字列を入力します。暗号アルゴリズムには aes (AES128) または bf (Blowfish) を選択できます。cipher_key_regist.sh スクリプトは TDE を利用するデータベース単位に実行する必要があります。

表 4　入力項目

項目	プロンプト	備考
ポート番号	Please enter database server port to connect	
ユーザー名	Please enter database user name to connect	管理者権限
パスワード	Please enter password for authentication	非表示
データベース名	Please enter database name to connect	
暗号鍵	Please enter the new cipher key	非表示
暗号鍵再入力	Please retype the new cipher key	非表示
暗号アルゴリズム	Please enter the algorithm for new cipher key	
確認	Are you sure to register new cipher key(y/n)	

例 7 暗号鍵の設定

```
$ cd $TDEHOME/SOURCES
$ sh bin/cipher_key_regist.sh $PGHOME
=== Database connection information ===
Please enter database server port to connect : 5432
Please enter database user name to connect : postgres
Please enter password for authentication : <<非表示>>
Please enter database name to connect : postgres
=== Regist new cipher key ===
Please enter the new cipher key : <<非表示>>
Please retype the new cipher key : <<非表示>>
Please enter the algorithm for new cipher key : aes

Are you sure to register new cipher key(y/n) : y
cipher_key_enable_log
----------------------
 t
(1 row)
```

暗号鍵（cipher key）に指定する文字は、オンライン・マニュアル上には以下のように記述されていますが、チェックは行われていないようです。

・{「!」,「'」}の同時使用,「¥」,「'」,「"」の使用を禁止しています。

暗号鍵の設定が完了すると、cipher_key_regist.sh コマンドに指定したデータベース・ユーザーが所有する cipher_key_table テーブル[1]が作成されます。このテーブルには暗号化鍵とアルゴリズム名が保存されています。

表 5　cipher_key_table テーブルの構成

列名	データ型	内容
key	BYTEA	暗号化鍵
algorithm	TEXT	暗号化アルゴリズム名

例 8 鍵保存テーブルの確認

```
postgres=> ¥d
             List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | cipher_key_table | table | postgres
 (1 row)

postgres=> SELECT * FROM cipher_key_table ;
                     key                                      | algorithm
--------------------------------------------------------------+-----------
¥xc30c040901024a2d3956dcbefa7dd2350190f9db21ad80d768891eee24f16250253cb8c02a947779
49b2a5ac69a6cb8060cd129fdebb765f5e5c2e15eeb89e01c774f9c631   | aes
(1 row)
```

key 列の値は、暗号化アルゴリズムにソルト付きの AES 256 を使って暗号化されて保存されています。

algorithm 列の値は暗号化／復号処理が行われる際に参照されます。このため、データが格納された後、この列を強制的に更新すると復号が正しく行えなくなります。

[1] public スキーマに作成されます。

例 9 鍵保存テーブルの強制変更

```
postgres=# UPDATE cipher_key_table set algorithm='bf' ; ← 異なる値に更新
UPDATE 1

postgres=> SELECT pgtde_begin_session('key1') ;  ← キーのオープン
 pgtde_begin_session
---------------------
 t
(1 row)
postgres=> SELECT * FROM tde1 ;
 col1 |          col2
------+-------------------------
 A    | *リm¥x14¥x16ｺｵMﾆﾓU胚  ←  復号が正しく行われない
(1 row)
```

2.3 利用方法

2.3.1 暗号化列の指定

TDE ではデータの暗号化は列単位に行うことができます。暗号化を行う列は、データ型として ENCRYPT_TEXT 型または ENCRYPT_BYTEA 型を指定してテーブルを作成します。テーブル作成時には暗号鍵のオープンは不要です。

表 6　暗号化データ型

データ型名	格納データ	STORAGE	備考
ENCRYPT_TEXT	テキスト・データ	extended	TEXT 相当
ENCRYPT_BYTEA	バイナリ・データ	extended	BYTEA 相当

例 10 テーブルの作成

```
$ psql -d postgres -U demo
postgres=> CREATE TABLE tde1 (col1 VARCHAR(10), col2 ENCRYPT_TEXT) ;
CREATE TABLE
postgres=> ¥d+ tde1
                              Table "public.tde1"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 col1   | character varying(10) |           | extended |              |
 col2   | encrypt_text          |           | extended |              |
Has OIDs: no
```

データの操作は通常の TEXT 型と同様ですが、データにアクセスする前に暗号鍵の利用開始処理が必要です。暗号鍵の有効化方法の詳細は後述しています。

例 11 データの格納と暗号化鍵の有効化

```
postgres=> INSERT INTO tde1 VALUES
        (generate_series(1, 10000)::text, generate_series(1, 10000)::text) ;
ERROR:  TDE-E0016 could not encrypt data, because key was not set(01)

postgres=> SELECT pgtde_begin_session('key1') ;
 pgtde_begin_session
---------------------
 t
(1 row)

postgres=> INSERT INTO tde1 VALUES
        (generate_series(1, 10000)::text, generate_series(1, 10000)::text) ;
INSERT 0 10000
```

2.3.2 暗号鍵の利用開始

TDE の機能を使う前に、pgtde_begin_session 関数に暗号鍵を指定して実行します。成功すると true が返ります。暗号鍵の登録時の指定と異なる文字列を入力するとエラーになります。

例 12 暗号鍵の利用開始

```
$ psql -d postgres -U demo
postgres=> SELECT pgtde_begin_session('badkey') ;
ERROR:  TDE-E0012 cipher key is not correct(01)
postgres=>
postgres=> SELECT pgtde_begin_session('key1') ;
 pgtde_begin_session
---------------------
 t
(1 row)
```

暗号鍵をオープンしない状態で、暗号化対象列にアクセスする場合や WHERE 句に暗号化対象列を指定するとエラーになります。

例 13 鍵をオープンしない状態の SQL（暗号化列にアクセスする場合）

```
$ psql -d postgres -U demo
postgres=> ¥d tde1 ;
        Table "public.tde1"
 Column |     Type     | Modifiers
--------+--------------+-----------
 col1   | text         |
 col2   | encrypt_text |
postgres=> SELECT * FROM tde1 ;
ERROR:  TDE-E0017 could not decrypt data, because key was not set(01)
postgres=> SELECT col1 FROM tde1 WHERE col2='DEF' ;
ERROR:  TDE-E0016 could not encrypt data, because key was not set(01)
LINE 1: SELECT col1 FROM tde1 WHERE col2='DEF';
                                         ^
postgres=> UPDATE tde1 SET col1='ABC' WHERE col2='DEF' ;
ERROR:  TDE-E0016 could not encrypt data, because key was not set(01)
LINE 1: UPDATE tde1 SET col1='ABC' WHERE col2='DEF' ;
                                               ^
postgres=> DELETE FROM tde1 WHERE col2 = 'DEF' ;
ERROR:  TDE-E0016 could not encrypt data, because key was not set(01)
LINE 1: DELETE FROM tde1 WHERE col2 = 'DEF' ;
                                      ^
postgres=> INSERT INTO tde1 VALUES ('ABC', 'DEF') ;
ERROR:  TDE-E0016 could not encrypt data, because key was not set(01)
LINE 1: INSERT INTO tde1 VALUES ('ABC', 'DEF') ;
                                        ^
```

暗号化対象以外の列のみにアクセスする SQL 文や復号が不要な SQL 文は、暗号鍵のオープンをしなくても実行可能です。検索だけでなく、UPDATE 文や DELETE 文も WHERE 句に暗号化対象列が含まれない場合は実行できます。

例 14 鍵をオープンしない状態の SQL（暗号化列にアクセスしない場合）

```
$ psql -d postgres -U demo
postgres=> SELECT COUNT(col1) FROM tde1 ;
 count
-------
 10000
(1 row)
postgres=> INSERT INTO tde1(col1) VALUES ('ABC') ;
INSERT 0 1
postgres=> UPDATE tde1 SET col1='XYZ' WHERE col1='ABC' ;
UPDATE 1
postgres=> DELETE FROM tde1 WHERE col1='XYZ' ;
DELETE 1
postgres=> TRUNCATE TABLE tde1 ;
TRUNCATE TABLE
postgres=> ALTER TABLE tde1 ADD COLUMN col3 ENCRYPT_TEXT ;
ALTER TABLE
postgres=> UPDATE tde1 SET col3 = col2 ; ←  復号不要のため実行可
UPDATE 1
```

2.3.3 インデックス

TDE より暗号化された列に対してインデックスの作成を検証しました。一般的に暗号化を行うと元のデータとバイト順序が変更されるため、BTREE インデックスによる範囲検索は実行できません。また復号されたデータによるインデックス作成は情報漏えいの恐れがあるため行われません。このため TDE では HASH インデックスのみサポートされています。インデックス作成時には独自の演算子クラスが使用されます。インデックス作成時には暗号鍵のオープンは不要です。

例 15 インデックスの作成と実行計画

```
postgres=> CREATE INDEX idx1_tde1 ON tde1 USING hash (col2) ;
CREATE INDEX
postgres=> EXPLAIN SELECT * FROM tde1 WHERE col2 = '1000' ;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using idx1_tde1 on tde1  (cost=0.00..8.02 rows=1 width=15)
   Index Cond: (col2 = '1000'::encrypt_text)
(2 rows)
```

表 7　独自演算子クラス

演算子クラス	説明	対応データ型
hashtext_enc_ops	暗号化されたテキスト列に使用	ENCRYPT_TEXT
hashbytea_enc_ops	暗号化されたバイナリ列に使用	ENCRYPT_BYTEA

2.3.4 パラメーター

TDE には以下の独自パラメーターが定義されています。

表 8　独自パラメーター

ツール	説明	データ型	デフォルト
encrypt.enable	自動暗号化／復号を行う	boolean	on
encrypt.checklogparam	log_statement のチェックを行う	boolean	on
encrypt.backup	暗号鍵のバックアップ先ディレクトリ名	text	''

□　パラメーターencrypt.enable

パラメーターencrypt.enable は暗号化／復号の機能をオフに設定します。このパラメーターを off に設定すると、暗号化列の検索では復号が行われません。また更新処理を行った場合でも自動暗号化が行われません。

例 16 パラメーターencrypt.enable = off

```
postgres=> SHOW encrypt.enable ;
 encrypt.enable
----------------
 off
(1 row)

postgres=> SELECT * FROM tde1 ;
 col1  |                  col2
-------+----------------------------------
 1     | ¥x01009744d18e8d119abb75b3a3006bddf7da
 2     | ¥x0100ef24b0b5ca1d13443d2f2bda490a560d
 3     | ¥x0100ebea69d6f8c21a24cbb680b504244eec
 4     | ¥x010039a022be21f4af5a850a1640c1888b69
<<以下省略>>
```

パラメーターencrypt.enable を off に設定すると、更新データは暗号化されずに格納されます。このパラメーターを on に戻し、暗号化データと非暗号化データが混在したテーブルを検索するとコネクションが異常終了する場合があることを確認しました。

例 17 暗号化／非暗号化混在テーブルの検索

```
-- パラメーターencrypt.enable = off に設定
-- tde1 テーブルに非暗号化データを INSERT
-- パラメーターencrypt.enable = on に設定し、インスタンス再起動
postgres=> SELECT pgtde_begin_session('key1') ;
 pgtde_begin_session
---------------------
 t
(1 row)
postgres=> SELECT * FROM tde1 ;
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Failed.
!>
```

2.3.5 スクリプトと関数

TDE には以下のスクリプト、関数が提供されます。

表 9　TDE が提供する主なスクリプト／関数

名称	説明
cipher_setup.sh	TDE のインストールを行うスクリプト
cipher_key_regist.sh	暗号化鍵の登録／更新を行うスクリプト
cipher_key_disable_log	ログ出力無効化ファンクション
cipher_key_enable_log	ログ出力有効化ファンクション
cipher_key_backup[2]	暗号鍵バックアップ・ファンクション
pgtde_begin_session	TDE セッション開始ファンクション
pgtde_end_session	TDE セッション終了ファンクション
cipher_key_regist	暗号鍵登録ファンクション[3]
cipher_key_reencrypt_data	暗号鍵変更による再暗号化[4]
enctext_*	テキスト内部／外部変換ファンクション
encbytea_*	テキスト内部／外部変換ファンクション
enc_compeq_*	データの比較を行うファンクション
enc_hash_*	ハッシュ値の計算を行うファンクション
enc_store_key_info	メモリ中に鍵情報を保存するファンクション
enc_store_old_key_info	メモリ中に古い鍵情報を保存するファンクション
enc_drop_key_info	メモリ中から鍵情報削除ファンクション
enc_drop_old_key_info	メモリ中から古い鍵情報削除ファンクション
enc_rename_backupfile	バックアップした暗号化鍵ファイルの名前変更ファンクション
enc_save_logsetting	ログ出力設定の退避ファンクション
enc_restore_logsetting	ログ出力設定のリストアファンクション
hashtext_enc_ops	ハッシュ・インデックスで利用
hashbytea_enc_ops	ハッシュ・インデックスで利用

[2] 要管理者権限

[3] cipher_key_regist.sh スクリプトから実行される。

[4] cipher_key_regist.sh スクリプトから実行される。

2.3.6 制約

TDE を利用するテーブル、列には以下の制約があります。

□ 暗号化列に対する BTREE インデックス作成不可

暗号化対象列には BTREE インデックス／一意インデックス（UNIQUE INDEX）を作成することができません。一意インデックスが作成できないため、主キー制約（PRIMARY KEY）や一意制約（UNIQUE）も作成できません。

例 18 BTREE インデックス／制約の作成

```
postgres => CREATE INDEX idx1_tde1 ON tde1(col2) ;
ERROR:  data type encrypt_text has no default operator class for access method
"btree"
HINT:  You must specify an operator class for the index or define a default
operator class for the data type.

postgres=> ALTER TABLE tde1 ADD CONSTRAINT pk_tde1 PRIMARY KEY (col2) ;
ERROR:  data type encrypt_text has no default operator class for access method
"btree"
HINT:  You must specify an operator class for the index or define a default
operator class for the data type.

postgres=> ALTER TABLE tde1 ADD CONSTRAINT uk_tde1 UNIQUE (col2) ;
ERROR:  data type encrypt_text has no default operator class for access method
"btree"
HINT:  You must specify an operator class for the index or define a default
operator class for the data type.
```

□ ログ出力設定

標準ではパラメーターlog_statement を'all'に設定した状態では、暗号鍵を有効化できません。この動作はパラメーターencrypt.checklogparam を off に設定することで回避できます（管理者のみ変更可能）。

標準では失敗した pgtde_begin_session 関数の実行状況はログに出力されますが、事前に cipher_key_disable_log 関数を実行することで pgtde_begin_session 関数のログ出力を停止することができます。

例 19 パラメーターlog_statement

```
postgres=> show log_statement ;
 log_statement
---------------
 all
(1 row)

postgres=> SELECT pgtde_begin_session('key1') ;
ERROR:  TDE-E0001 log_statement must not be 'all'(01)

$ tail data/pg_log/postgresql-2015-06-08_024732.log
LOG:  statement: SELECT pgtde_begin_session('key1') ;
ERROR:  TDE-E0001 log_statement must not be 'all'(01)
STATEMENT:  SELECT pgtde_begin_session('key1') ;
```

□ pg_dump / pg_dumpall コマンド

暗号鍵のオープン処理が実行できないため、pg_dump および pg_dumpall コマンドによるバックアップは失敗します。pg_basebackup コマンドは問題なく実行できます。

例 20 pg_dump の失敗

```
$ pg_dump -d postgres -U postgres -f postgres.dmp
pg_dump: Dumping the contents of table "tde1" failed: PQgetResult() failed.
pg_dump: Error message from server: ERROR:  TDE-E0017 could not decrypt data,
because key was not set(01)
pg_dump: The command was: COPY public.tde1 (col1, col2) TO stdout;
```

□ CHECK 制約と TEMPORARY TABLE

オンライン・マニュアルには CHECK 制約や TEMPORARY TABLE はサポート対象外となっていますが、検証した限りでは動作しています。サポートされない理由については不明です。

例 21 CHECK 制約と TEMPORARY TABLE

```
postgres=> CREATE TABLE tde2 (col1 TEXT, col2 ENCRYPT_TEXT
        CHECK(length(col2) > 3)) ;
CREATE TABLE
postgres=> INSERT INTO tde2 VALUES ('ABC', 'DEF') ;
ERROR:  new row for relation "tde2" violates check constraint "tde2_col2_check"
DETAIL:  Failing row contains (ABC, DEF).
postgres=> INSERT INTO tde2 VALUES ('ABC', 'DEFGHI') ;
INSERT 0 1

postgres=> CREATE TEMPORARY TABLE tde3 (col1 TEXT, col2 ENCRYPT_TEXT) ;
CREATE TABLE
postgres=> INSERT INTO tde3 VALUES ('ABC', 'DEF') ;
INSERT 0 1
```

□ TEXT 型との互換性

ENCRYPT_TEXT 型は基本的に標準の TEXT 型と互換性がありますが、CHAR/VARCHAR 型と直接連結する「||」オペレーターが実装されていないため、以下の構文が実行できません（col2 列は ENCRYPT_TEXT 型）。TEXT 型にキャストする必要があります。

例 22 文字列型との直接連結

```
postgres=> SELECT col2 || 'test' FROM tde1 ;
ERROR:  operator is not unique: encrypt_text || unknown
LINE 1: SELECT col2 || 'test' FROM tde1;
                    ^
HINT:  Could not choose a best candidate operator. You might need to add explicit
type casts.
postgres=> SELECT col2::text || 'test' FROM tde1 ;
?column?
----------
 DEFtest
(1 row)
```

上記以外の制約もオンライン・マニュアルに記載されていますが、検証は行っていません。

2.3.7 暗号鍵の管理

暗号鍵の管理と影響について検証しました。

□ バックアップ

暗号鍵のバックアップは、データベース管理者権限で cipher_key_backup 関数を実行します。内部では COPY 文により cipher_key_table テーブルの内容がファイルに出力されます。バックアップ先は、パラメーターencrypt.backup にバックアップ・データの出力先ディレクトリを指定します。バックアップ・データとして「ck_backup_{データベース名}」の名前でファイルが出力されます。

表 10　encrypt.backup パラメーターの指定

指定値	動作
省略	データベース・クラスタに出力
存在しないディレクトリ	エラー
書き込み権限が無いディレクトリ	エラー
同一ファイル名が存在するディレクトリ	既存ファイルを拡張子「.sv」をつけて退避し、新規ファイルを作成

例 23 暗号鍵のバックアップ

```
postgres=# SET encrypt.backup TO '/home/postgres' ;
SET
postgres=# SELECT cipher_key_backup() ;
 cipher_key_backup
-------------------
 t
(1 row)

$ ls -l ck*
-rw-r--r--. 1 postgres postgres 145 Jun  8 06:51 ck_backup_postgres
-rw-r--r--. 1 postgres postgres 145 Jun  8 06:49 ck_backup_postgres.sv
```

cipher_key_backup 関数は、暗号鍵の登録を行う cipher_key_regist.sh スクリプト実行時にも自動的に動作します。

□ 暗号鍵の変更

暗号鍵の変更は cipher_key_regist.sh スクリプトを再実行します。以下の入力項目が必要です。

表 11 入力項目

項目	プロンプト	備考
ポート番号	Please enter database server port to connect	
ユーザー名	Please enter database user name to connect	管理者権限
パスワード	Please enter password for authentication	非表示
データベース名	Please enter database name to connect	
現在の暗号鍵	Please enter the current cipher key	非表示
新規暗号鍵	Please enter the new cipher key	
暗号鍵再入力	Please retype the new cipher key	非表示
暗号アルゴリズム	Please enter the algorithm for new cipher key	
確認	Are you sure to register new cipher key(y/n)	

暗号化鍵を変更すると既存テーブル内の暗号化対象列はすべて再暗号化されます。

例 24 暗号鍵の再設定

```
$ sh bin/cipher_key_regist.sh $PGHOME
=== Database connection information ===
Please enter database server port to connect : 5432
Please enter database user name to connect : postgres
Please enter password for authentication : <<非表示>>
Please enter database name to connect : postgres
=== Regist new cipher key ===
Please enter the current cipher key : <<非表示>>
Please enter the new cipher key : <<非表示>>
Please retype the new cipher key : <<非表示>>
Please enter the algorithm for new cipher key : bf

Are you sure to register new cipher key(y/n) : y
INFO:  TDE-I0001 re-encryption of table "public"."tde1" was started(01)
CONTEXT:  SQL statement "SELECT cipher_key_reencrypt_data(current_cipher_key,
current_cipher_algorithm, cipher_key)"
PL/pgSQL function cipher_key_regist(text,text,text) line 65 at PERFORM
INFO:  TDE-I0002 re-encryption of table "public"."tde1" was completed(01)
CONTEXT:  SQL statement "SELECT cipher_key_reencrypt_data(current_cipher_key,
current_cipher_algorithm, cipher_key)"
<<途中省略>>
PL/pgSQL function cipher_key_regist(text,text,text) line 65 at PERFORM
 cipher_key_enable_log
-----------------------
 t
(1 row)

$
```

2.4 Oracle Database との比較

Oracle Database には TDE に相当する機能が搭載されています。Oracle Database Enterprise Edition のオプション機能である Oracle Advanced Security を購入すると利用できます。ここでは Oracle Database の TDE との比較を行っています。

表 12 Oracle Database との比較

比較項目	Oracle Database	PostgreSQL
無償版	なし	あり
暗号化メソッドの設定単位	列	データベース
暗号化メソッド	AES128 AES192 AES256 3DES168	AES128 Blowfish
整合性指定	可能	不可
ソルトの指定	可能	不可
暗号化データ型	標準データ型 - 文字列型 - LOB - NUMBER - DATE - RAW	独自データ型 - ENCRYPT_TEXT - ENCRYPT_BYTEA
暗号鍵のオープン	インスタンス単位	セッション単位
列暗号化	可能	可能
表領域暗号化	可能	不可
暗号化鍵の保存場所	ファイル, PKCS#11 HSM	テーブル
暗号化鍵の更新	可能	可能
鍵更新時の動作	なし	再暗号化
暗号化鍵のバックアップ	ファイル操作	専用関数
インデックスの作成	BTREE 可能	HASH のみ
主キーの暗号化	可能	不可

□ リスク対応の比較

Oracle Database の TDE は暗号鍵の有効化がインスタンス単位であるため、データベースに接続するユーザー名とパスワードが漏洩した場合には暗号化データにアクセスできてしまいます。PostgreSQL はセッション単位に暗号鍵のオープン処理を行うため、鍵情報が

漏洩しない限り暗号化データにアクセスできません。

一方で TDE for PostgreSQL では、暗号化列にアクセスする際には暗号鍵を指定した関数を実行する必要があります。このため暗号鍵の情報を PostgreSQL インスタンスにアクセスするクライアント上で保持する必要があります。Oracle Database では暗号化鍵のオープン処理はインスタンス単位で行われるためクライアントに鍵情報を保持する必要がありません。

3. まとめと感想

Transparent Data Encryption for PostgreSQL は PostgreSQL に自動暗号化／復号機能を提供しています。PCIDSS 等のセキュリティ要件に対応するために暗号化は必須の機能であり、既存アプリケーションの修正を最小限に抑えることができる TDE は非常に有効なソリューションであると思われます。

実装としては、暗号化対象として新しいデータ型を提供し、データ型に対する入出力関数を定義することで自動的に暗号化／復号する機能を実現しています。この実装方法により、PostgreSQL 本体に手を入れずに機能を実現することに成功しています。暗号化／復号処理自体には PostgreSQL に標準搭載されている pgcrypto モジュールを使用しているため、暗号の強度や実装に対する不安を小さくしています。

以下の点が改善すると更に使い勝手が良くなると思われます。

- BTREE インデックス
  現状では HASH インデックスのみ利用できますが、BTREE インデックスをサポートして欲しいと思います。BTREE インデックスがサポートされると、主キーとして使えるようになるため、カード番号等にも利用が可能になります。

- 最新バージョンへの対応
  PostgreSQL 9.4 への対応を希望します。

- '||' オペレーターの実装
  ENCRYPT_TEXT 型と文字列連結を行う || オペレーターを実装して欲しいと思います。

- インスタンス単位の暗号化鍵有効化
  現状ではセッション単位に pgtde_begin_session 関数を実行する必要があります。インスタンス単位で暗号鍵をオープンできる仕組みがあれば既存アプリケーションへの修正が不要になります。

- ドキュメントの充実
  CHECK 制約や TEMPORARY TABLE がサポート対象外である理由や、パラメーター設定時の動作等についてドキュメントの充実を希望します。

変更履歴

変更履歴

版	日付	作成者	説明
1.0	8-June-2015	篠田典良	社内版作成
1.0.1	15-June-2015	篠田典良	誤字修正 公開版作成

以上