



EDB

Postgres® for the AI Generation

Transparent Data Encryption

リリース *November 2024*

EnterpriseDB Corporation

2024 年 11 月 20 日

目次

第 1 章 Overview	3
1.1 About TDE	3
1.1.1 TDE はパフォーマンスにどのような影響を与えますか?	3
1.2 TDE compatibility	6
1.2.1 サポートされているデータベースディストリビューション	6
1.2.2 サポートされているテクノロジー	6
第 2 章 Administering	7
2.1 Overview	7
2.1.1 始める前に	7
2.1.2 サーバーの初期化	7
2.1.3 データの移行既存のインスタンスの場合	8
2.1.4 チュートリアル	8
2.2 Securing the data encryption key	8
2.2.1 データ暗号化キー	8
2.2.2 データ暗号化キーを保護するメカニズムの選択	9
2.2.3 ラッピングおよびアンラッピングコマンドの構成	9
2.2.4 TDE にラッピングおよびアンラップコマンドを提供する	9
2.3 Using TDE initialization options	15
2.3.1 TDE を有効にするには	15
2.3.2 データ暗号化キーを保護するには	16
2.3.3 オプションリファレンス	16
2.4 Working with encrypted files	16
2.4.1 Working with encrypted WAL files	16
2.4.2 Working with encrypted backup files	18
2.5 Upgrading a TDE-enabled database	19
2.5.1 チュートリアル	20
2.6 Tutorials	20
2.6.1 Creating a database with TDE	20
2.6.2 Enabling TDE on an existing EDB Postgres Advanced Server	21
2.6.3 Upgrading PostgreSQL to EDB Postgres Extended Server while enabling TDE	25
2.6.4 Verifying TDE is enabled	28
第 3 章 Reference	29
3.1 Commands affected by TDE	29
3.2 initdb TDE options	29
3.2.1 オプション <code>--data-encryption</code> または <code>-y</code>	29
3.2.2 オプション <code>--key-wrap-command=<command></code>	30
3.2.3 オプション <code>--key-unwrap-command=<command></code>	30
3.2.4 オプション <code>--no-key-wrap</code>	30

3.2.5	オプション <code>--copy-key-from=<file></code>	31
3.3	<code>pg_upgrade</code> TDE options	31
3.3.1	オプション <code>--copy-by-block</code>	31
3.3.2	オプション <code>--key-unwrap-command=<command></code>	32
3.4	Limitations	32
3.4.1	<code>FILE_COPY</code>	32

透過的データ暗号化 TDE は、バージョン 15 から EDB EDB Postgres Advanced Server および EDB Postgres Extended Server でサポートされているオプション機能です。

データベースシステムに保存されているユーザーデータを暗号化します。この暗号化はユーザーに対して透過的です。ユーザーデータには、テーブルやその他のオブジェクトに格納されている実際のデータ、およびオブジェクトの名前などのシステムカタログデータが含まれます。

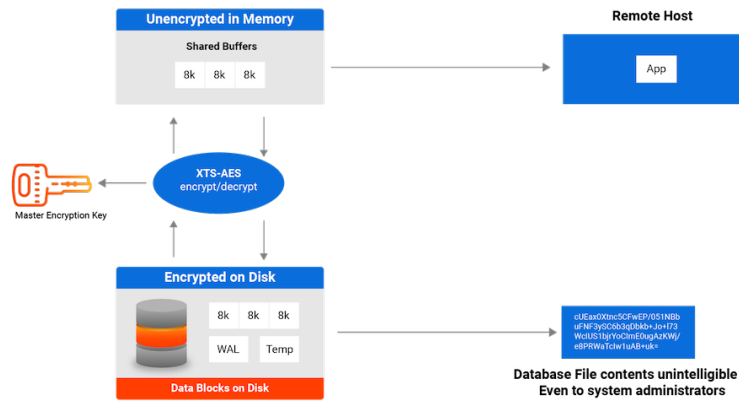


図 1: Architecture diagram

第1章 Overview

1.1 About TDE

1.1.1 TDE はパフォーマンスにどのような影響を与えますか？

TDE のパフォーマンスへのインパクトは低いです。詳細については、[Transparent Data Encryption Impacts on EDB Postgres Advanced Server 15](#) ブログを参照してください。

透過的データ暗号化 TDE は、バージョン 15 から EDB EDB Postgres Advanced Server および EDB Postgres Extended Server でサポートされているオプション機能です。

データベースシステムに保存されているユーザーデータを暗号化します。

How does TDE encrypt data?

TDE は、データベースサーバーおよびバックアップストレージ上のオペレーティングシステムファイルのデータの不正な表示を防ぎます。データは、盗難または置き忘れられた場合、未承認のユーザーが理解できなくなります。

データの暗号化と復号化はデータベースによって管理され、アプリケーションの変更やクライアントドライバの更新は必要ありません。

EDB Postgres Advanced Server および EDB Postgres Extended Server は、データベースの外部のキー管理へのフックを提供します。これらのフックにより、単純なパスフレーズの暗号化/復号化、またはエンタープライズキー管理ソリューションとの統合が可能になります。詳細については、[Securing the data encryption key](#) を参照してください。

TDE はパフォーマンスにどのような影響を与えますか？

TDE のパフォーマンスへのインパクトは低いです。詳細については、[Transparent Data Encryption Impacts on EDB Postgres Advanced Server 15](#) ブログを参照してください。

TDE はどのようにデータを暗号化しますか？

EDB TDE は [OpenSSL](#) を使用して、AES 暗号化アルゴリズムでデータファイルを暗号化します。Windows システムでは、TDE は [OpenSSL 3](#) を使用します。Linux システムでは、TDE はホストオペレーティングシステムにインストールされている OpenSSL バージョンを使用します。インストールされたバージョンを確認するには、`openssl version` を実行します。詳細については、[OpenSSL documentation](#) を参照してください。OpenSSL コミュニティが提供していないカスタムビルドを使用している場合は、ベンダーのドキュメントを参照してください。

バージョン 16 以降、EDB TDE は、Postgres クラスターの初期化中に AES-128 および AES-256 暗号化アルゴリズムを選択するオプションを導入します。AES-128 と AES-256 の選択は、パフォーマンスとセキュリティ要件のバランス次第です。AES-128 は、一般的にパフォーマンス効率と低消費電力が重要な環境に推奨され、ほとんどのアプリケーションに適しています。逆に、AES-256 は、最高レベルのセキュリティが必要なシナリオ、多くの場合規制上の義務によって駆動される場合にお勧めします。

TDE は、データファイルの暗号化に AES-128-XTS または AES-256-XTS アルゴリズムを使用します。XTS は、調整値と呼ばれる 2 番目の値を使用して、暗号化を強化します。TDE を使用した XTS 調整値は、データベース OID、relfile ノード、およびブロック番号を使用します。

ログ先行書き込み WAL ファイルの場合、TDE は AES-128-CTR または AES-256-CTR を使用し、WAL のログシーケンス番号 LSN をカウンタコンポーネントとして組み込みます。

ブロックによってアクセスされる一時ファイルも、AES-128-XTS または AES-256-XTS を使用して暗号化されます。他の一時ファイルは、AES-128-CBC または AES-256-CBC を使用して暗号化されます。

TDE ではデータはどのようにディスクに保存されますか？

この例では、`tbfoo` テーブルのデータが暗号化されます。`pg_relation_filepath` ファンクションは、`tbfoo` テーブルに対応するデータファイルを見つけます。

```
insert into tbfoo values (abc,123);
INSERT 0 1

select pg_relation_filepath(tbfoo);

pg_relation_filepath
-----
base/5/16416
```

データを `grep` して文字を検索しても、何も返されません。最後の 5 行を表示すると、暗号化されたデータが返されます。

```
$ hexdump -C 16416 | grep abc
$

$ hexdump -C 16416 | tail -5
00001fc0 c8 0f 1d c8 9a 63 3d dc 7d 4e 68 98 b8 f2 5e 0a |.....c=.]Nh...^.|
00001fd0 9a eb 20 1d 59 ad be 94 6e fd d5 6e ed 0a 72 8c |.. .Y...n..n..r.|
```

(次のページに続く)

(前のページからの続き)

```

00001fe0 7b 14 7f de 5b 63 e3 84 ba 6c e7 b0 a3 86 aa b9 |{...[c...l.....|
00001ff0 fe 4f 07 50 06 b7 ef 6a cd f9 84 96 b2 4b 25 12 |.O.P...j.....K%.|
00002000

```

What's encrypted with TDE?

TDE は以下を暗号化します

- TOAST テーブルとシステムカタログを含む、すべてのフォークを含むテーブル、シーケンス、インデックスの基になるファイル。これらのファイルは**データファイル**と呼ばれます。
- ログ先行書き込み WAL。
- クエリ処理およびデータベースシステムの操作中に使用されるさまざまな一時ファイル。

以下は TDE によって暗号化されず、その他の方法で偽装されません。

- トランザクションステータス `pg_subtrans`、`pg_xact` などのユーザーデータを含まないデータベースシステムの動作に関する内部メタデータたとえば、`pg_subtrans`、`pg_xact`。
- データディレクトリ内のファイル名とファイルシステム構造。これは、データベースシステムの全体のサイズ、データベースの数、テーブルの数、それらの相対サイズ、および最終アクセス時間などのファイルシステムのメタデータがすべて復号化せずに可視であることを意味します。
- 外部テーブルのデータ。
- サーバー診断ログ。
- 構成ファイル。

Why should you use TDE?

TDE 暗号化により、ユーザーデータは不正なアクセスから保護されます。

`data encryption key securing mechanism` で構成されている場合、データベースサーバーとバックアップに保存されているデータは、復号キーを使用してユーザーとプロセスのみがアクセスできます。

一部の使用例には次のものが含まれます。

- **機密性の高い個人データの保護** 金融、電子商取引、ヘルスケア、政府機関などの業界は、GDPR、HIPPA、PCI DSS などのデータプライバシー規制に準拠するために保護する必要がある個人識別情報を扱うことがよくあります。
- **政府標準へのコンプライアンス** 政府機関は、FIPS のような情報処理標準に準拠して、コンピューターのセキュリティと相互運用性を保証する必要があります。
- **取引データの保護** 金融機関は、詐欺や経済的損失を防ぐために保護する必要がある取引、口座、および支払いデータを扱います。
- **知的財産の保護** 組織は、競争上の優位性を維持し、ブランド価値をサポートし、イノベーションを促進するために、独自の情報、デザイン、および計画を保護します。

- クラウドベースの展開およびパブリック Web アプリケーションにおけるデータの保護データベースのデータの暗号化は、インフラストラクチャが共有されている場合、またはアプリケーションの API に脆弱性が侵入する可能性がある場合に、追加のセキュリティレイヤーを提供します。

データが暗号化されると、盗難または置き忘れた場合に理解できなくなります。

1.2 TDE compatibility

いくつかのデータベースディストリビューションとテクノロジーで TDE 対応データベースサーバーを作成できます。

1.2.1 サポートされているデータベースディストリビューション

次の場所に TDE 対応データベースを作成できます。

- EDB Postgres Advanced Server 15 以降のバージョン。
- EDB Postgres Extended Server 15 以降のバージョン。

注釈: EDB は、PostgreSQL での TDE 対応サーバーの作成をサポートしていません。TDE は、EDB Postgres Advanced および Extended Servers 用に EDB が開発したメカニズムに依存しています。!!!

1.2.2 サポートされているテクノロジー

次の環境で TDE 対応サーバーを作成できます。

- EDB Postgres Distributed (PGD)
- EDB Postgres Distributed for Kubernetes
- EDB Postgres for Kubernetes
- EDB Postgres AI Cloud Service

第2章 Administering

2.1 Overview

データベースで透過的データ暗号化 TDE の使用を開始する場合は、新しい TDE 対応データベースサーバーを作成するか、既存のデータベースサーバーを TDE 対応環境に移行します。既存のインスタンスで TDE を有効にすることはできません。

データベースサーバーをスクラッチから作成するか、新しいインスタンスを作成して既存のデータベースサーバーを移行するかに関係なく、`initdb` を使用してデータベースクラスターを初期化することにより、TDE 対応データベースを作成する必要があります。

2.1.1 始める前に

- TDE によって生成された `secure the data encryption key` の方法を選択します。 *Using a passphrase*、*Using a key store* からのラッピングキーでそれらのキーを保護するか、テストのために *Disabling the key wrapping* を選択できます。
- *initdb TDE options* を確認して、TDE 対応データベースクラスターの初期化に必要なすべての情報を持っていることを確認します。
- *Limitations* を確認して、TDE を有効にした場合の PostgreSQL ユーティリティの処理の制限と変更を理解します。
- 既存のデータベースサーバーからデータを移行する予定がある場合は、ソースデータベースサーバーのバックアップを必ず実行してください。

2.1.2 サーバーの初期化

1. Wrapping および unwrapping コマンドをエクスポートして、暗号化キーを保護します。計画段階で選択したラッピング方法を使用します。

または、コマンドライン引数でサーバーを初期化するときに、ラッピングおよびアンラップコマンドを提供できます。

例については、*TDE にラッピングおよびアンラップコマンドを提供する* を参照してください。

1. ターゲットディレクトリで `--data-encryption` を有効にしてデータベースサーバーを初期化します。必要に応じて、他の *initdb TDE options* を含めます。
2. データベースクラスターと *Verifying TDE is enabled* を起動します。

詳細な初期化例については、[Tutorials](#) を参照してください。

2.1.3 データの移行既存のインスタンスの場合

既存のデータベースサーバーからデータとオブジェクトを移行する場合は、`pg_upgrade` を使用して既存のインスタンスからデータをコピーします。

1. ソースサーバーと新しいサーバーの両方を停止します。
2. `pg_upgrade` と `--copy-by-block` オプションを使用して、ソースサーバーから新しいサーバーにデータをコピーします。必要に応じて、他の `pg_upgrade TDE options` を含めます。
3. 新しい暗号化されたデータベースサーバーを起動します。
4. 暗号化されたデータベースサーバーに接続し、データが転送されたことを確認します。
5. 必要なクリーンアップ操作を実行します。

注釈: サポートされている有効化と移行のユースケースの概要については、[Upgrading a TDE-enabled database](#) を参照してください。

詳細な移行例については、[Tutorials](#) を参照してください。

2.1.4 チュートリアル

- [Creating a database with TDE](#) 。
- [Enabling TDE on an existing EDB Postgres Advanced Server](#) 。
- [Upgrading PostgreSQL to EDB Postgres Extended Server while enabling TDE](#) 。

2.2 Securing the data encryption key

2.2.1 データ暗号化キー

透過的データ暗号化のキーデータキーは `initdb` によって生成され、データディレクトリの下のファイル `pg_encryption/key.bin` に保存されます。このファイルには、実際には、実行時にさまざまな目的に使用される複数のキーが含まれています。データキーは、ファイル内のランダムなバイトの単一シーケンスです。

これ以上のアクションがなければ、このファイルにはプレーンテキストのキーが含まれていますが、安全ではありません。暗号化されたデータディレクトリにアクセスできる人は誰でも平文キーにアクセスできます。これは、暗号化の目的を無効にします。

2.2.2 データ暗号化キーを保護するメカニズムの選択

データ暗号化キーを保護するには、TDE にデータ暗号化保護メカニズムを提供する `wrap` および `unwrap` コマンドを指定する必要があります。このパラメーターは、サーバーの起動時にのみ設定できます。

`Wrap` および `unwrap` コマンドを使用すると、次のことができます。

- *Using a passphrase*。ラッピングキーはパスフレーズから取得され、データキーの暗号化に使用されます。
- *Using a key store* またはキーストア。この 2 番目のキーは、**キーラッピングキー** または **マスターキー** と呼ばれます。
- *Disabling the key wrapping* を選択することもできます。TDE キーが保護されないままになるため、これはテスト環境でのみ行ってください。

2.2.3 ラッピングおよびアンラッピングコマンドの構成

キーを保護する方法を選択したら、ラッピング/ラップ解除コマンドを作成できます。これらのコマンドの構成はユーザーに任されるため、セットアップをローカル要件に合わせて調整し、既存のキー管理ソフトウェアなどと統合できます。

Using a passphrase トピックでは、OpenSSL とパスフレーズを使用してコマンドをラッピング/アンラップする例を提供して TDE データキーを保護します。*Using a key store* トピックは、外部キーストアキーを使用してコマンドをラップ/アンラップして TDE データキーを保護する例を提供します。

TDE でサーバーを初期化すると、`initdb` コマンドは `postgresql.conf` 構成ファイルに `data_encryption_key_unwrap_command` パラメーターを追加します。`data_encryption_key_unwrap_command` で指定された文字列は、データ暗号化キーをラップ解除または復号化できます。

コマンドには、プレースホルダー `%p` が含まれる必要があります。これは、ラップ解除するキーを含むファイルの名前に置き換えられます。コマンドは、ラップされていない復号化されたキーを標準出力に出力する必要があります。

2.2.4 TDE にラッピングおよびアンラップコマンドを提供する

TDE データベースサーバーでコマンドを利用できるようにして、データ暗号化キーをラップおよびラップ解除できるようにする必要があります。次のオプションがあります。

- データベースを作成する前に、ラッピングおよびアンラップコマンドを環境変数として構成できるため、TDE はサーバーの初期化時にこれらの変数にフォールバックできます。

例

```
PGDATAKEYWRAPCMD=openssl enc -e -aes128-wrap -pbkdf2 -out "%p"
PGDATAKEYUNWRAPCMD=openssl enc -d -aes128-wrap -pbkdf2 -in "%p"
export PGDATAKEYWRAPCMD PGDATAKEYUNWRAPCMD
```

(次のページに続く)

(前のページからの続き)

#After these variables are set, you can initialize the server:

```
initdb --data-encryption -D /var/lib/edb/as16/data
```

- `--key-wrap-command=<command>` および `--key-unwrap-command=<command>` オプションを使用してサーバーを初期化するときに、コマンドラインで直接ラッピングおよびアンラップコマンドを提供できます。

例

```
initdb --data-encryption -D /var/lib/edb/as16/data --key-wrap-command=openssl enc -e -
→aes128-wrap -pbkdf2 -out "%p" --key-unwrap-command=openssl enc -d -aes128-wrap -
→pbkdf2 -in "%p"
```

- `--no-key-wrap` を使用してデータ暗号化キーの保護を無効にできます。これにより、キーは保護されないままになり、プロポーザルのテストにのみお勧めします。

例

```
initdb --data-encryption -D /var/lib/edb/as16/data --no-key-wrap
```

Using a passphrase

openssl コマンドラインユーティリティを使用して、パスフレーズでデータキーを保護できます。次に、この保護を設定する例を示します。

```
initdb -D datadir -y --key-wrap-command=openssl enc -e -aes-128-cbc -pbkdf2 -out "%p"
→--key-unwrap-command=openssl enc -d -aes-128-cbc -pbkdf2 -in "%p"
```

この例では、ランダムに生成されたデータキー initdb によって内部的に実行を、AES-128-CBC AESKW アルゴリズムで暗号化することによりラップします。暗号化には、PBKDF2 キー導出ファンクションを使用してパスフレーズから導出されたキーと、ランダムに生成されたソルトが使用されます。ターミナルはパスフレーズの入力を求めます。これらのオプションの詳細については、openssl-enc のマニュアルページを参照してください。使用可能なオプションはバージョンによって異なります。initdb ユーティリティは、%p をラップしたキーを保存するファイルの名前に置き換えます。

unwrap コマンドは、逆の操作を実行します。initdb は、unwrap 操作を必要としません。ただし、それは初期化されたクラスターの postgresql.conf ファイルに保存され、起動時に使用されます。

key Wrap コマンドは、標準入力でプレーンテキストキーを受け取り、%p プレースホルダーで指定されたファイルシステムの場所にラップされたキーを配置する必要があります。key unwrap コマンドは、%p プレースホルダーで指定されたファイルシステムの場所からラップされたキーを読み取り、ラップされていないキーを標準出力に書き込む必要があります。

pg_rewind や pg_upgrade などのユーティリティプログラムは、データディレクトリまたはバックアップなどのコピーを直接動作します。状況に応じて、これらのプログラムに key unwrap コマンドについて伝える必要があります。これらにはそれぞれ、この目的のためのコマンドラインオプションがあります。

操作を簡素化するために、環境変数にキー wrap および unwrap コマンドを設定することもできます。これらは、対応するコマンドラインオプションを提供しない場合、影響を受けるすべてのアプリケーションで受け入れられます。例

```
PGDATAKEYWRAPCMD=openssl enc -e -aes-128-cbc -pbkdf2 -out "%p"
PGDATAKEYUNWRAPCMD=openssl enc -d -aes-128-cbc -pbkdf2 -in "%p"
export PGDATAKEYWRAPCMD PGDATAKEYUNWRAPCMD
```

サーバーが pg_ctl または systemd などのサービスマネージャーを介して起動された場合、端末でパスワードの入力を求めるキーのラップ解除コマンドは機能しません。これらの環境では、サーバーはターミナルから切断されます。サーバー起動時に対話型パスワードプロンプトが必要な場合は、間接的なメカニズムを使用してパスワードを取得するより精巧な構成が必要です。

たとえば、systemd の場合、systemd-ask-password を使用できます。

```
PGDATAKEYWRAPCMD="bash -c openssl enc -e -aes-128-cbc -pbkdf2 -out %p -pass file:
↳<(sudo systemd-ask-password --no-tty)"
PGDATAKEYUNWRAPCMD="bash -c openssl enc -d -aes-128-cbc -pbkdf2 -in %p -pass file:
↳<(sudo systemd-ask-password --no-tty)"
```

/etc/sudoers のようなエントリーも必要です。

```
postgres ALL = NOPASSWD: /usr/bin/systemd-ask-password
```

Using a key store

外部キー管理システムのキーストアを使用して、データ暗号化キーを管理できます。テストおよびサポートされているキーストアは次のとおりです。

- Amazon AWS Key Management ServiceKMS
- Microsoft Azure Key Vault
- Google Cloud - Cloud Key Management Service
- HashiCorp Vault KMIP Secrets エンジンおよび Transit Secrets エンジン
- Thales CipherTrust Manager
- Fortanix データセキュリティマネージャー
- Entrust KeyControl

使用可能なキーストアのいずれかを使用するには、構成例を参照してください。

AWS Key Management Service の例

AWS Key Management Service でキーを作成します。

```
aws kms create-key
aws kms create-alias --alias-name alias/pg-tde-master-1 --target-key-id "..."
```

aws kms コマンドと alias/pg-tde-master-1 キーを使用して、データ暗号化キーをラップおよびラップ解除します。

```
PGDATAKEYWRAPCMD=aws kms encrypt --key-id alias/pg-tde-master-1 --plaintext fileb:///
→dev/stdin --output text --query CiphertextBlob | base64 -d > "%p"
PGDATAKEYUNWRAPCMD=aws kms decrypt --key-id alias/pg-tde-master-1 --ciphertext-blob
→fileb:///"%p" --output text --query Plaintext | base64 -d
```

注釈: この例のようなパイプを使用したシェルコマンドは、パイプの終了ステータスが最後のコマンドの終了ステータスであるため、問題があります。最初のより興味深いコマンドの失敗は、適切に報告されません。Postgres は、wrap または unwrap コマンドが何も書き込んだかどうかを認識することにより、これをある程度処理します。ただし、これをより堅牢にすることをお勧めします。たとえば、一部のシェルで使用可能な `pipefail` オプション、または一部のオペレーティングシステムで使用可能な `mispipe` コマンドを使用します。より複雑なコマンドをインラインで定義するのではなく、外部シェルスクリプトまたは他のプログラムに配置します。

Azure Key Vault の例

Azure Key Vault でキーを作成します。

```
az keyvault key create --vault-name pg-tde --name pg-tde-master-1
```

az keyvault key コマンドと pg-tde-master-1 キーを使用して、データ暗号化キーをラップおよびラップ解除します。

```
PGDATAKEYWRAPCMD=az keyvault key encrypt --name pg-tde-master-1 --vault-name pg-tde --
→algorithm A256GCM --value @- --data-type plaintext --only-show-errors --output json
→| jq -r .result > "%p"
PGDATAKEYUNWRAPCMD=az keyvault key decrypt --name pg-tde-master-1 --vault-name pg-tde
→--algorithm A256GCM --value @"%p" --data-type plaintext --only-show-errors --output
→json | jq -r .result
```

注釈: この例のようなパイプを使用したシェルコマンドは、パイプの終了ステータスが最後のコマンドの終了ステータスであるため、問題があります。最初のより興味深いコマンドの失敗は、適切に報告されません。Postgres は、wrap または unwrap コマンドが何も書き込んだかどうかを認識することにより、これをある程度処理します。ただし、これをより堅牢にすることをお勧めします。たとえば、一部のシェルで使用可能な `pipefail` オプション、または一部のオペレーティングシステムで使用可能な `mispipe` コマン

ドを使用します。より複雑なコマンドをインラインで定義するのではなく、外部シェルスクリプトまたは他のプログラムに配置します。

Google Cloud KMS の例

Google Cloud KMS でキーを作成します。

```
gcloud kms keys create pg-tde-master-1 --location=global --keyring=pg-tde --
↳purpose=encryption
```

gcloud kms コマンドと pg-tde-master-1 キーを使用して、データ暗号化キーをラップおよびラップ解除します。

```
PGDATAKEYWRAPCMD=gcloud kms encrypt --plaintext-file=- --ciphertext-file=%p --
↳location=global --keyring=pg-tde --key=pg-tde-master-1
PGDATAKEYUNWRAPCMD=gcloud kms decrypt --plaintext-file=- --ciphertext-file=%p --
↳location=global --keyring=pg-tde --key=pg-tde-master-1
```

HashiCorp Vault Transit Secrets Engine の例

HashiCorp Vault Transit Secrets Engine でトランジットを有効にします

```
vault secrets enable transit
```

キーを作成し、名前を付けます。

```
vault write -f transit/keys/pg-tde-master-1
```

vault write コマンドと pg-tde-master-1 キーを使用して、データ暗号化キーをラップおよびラップ解除します。

```
PGDATAKEYWRAPCMD=base64 | vault write -field=ciphertext transit/encrypt/pg-tde-master-
↳1 plaintext=- > %p
PGDATAKEYUNWRAPCMD=vault write -field=plaintext transit/decrypt/pg-tde-master-1
↳ciphertext=- < %p | base64 -d
```

Thales CipherTrust Manager の例

データ暗号化キーを Thales キーストアのキーでラップする方法については、Implementing Thales CipherTrust Manager ドキュメントの *Using TDE initialization options* セクションを参照してください。

Fortanix Data Security Manager の例

ステップバイステップの構成チュートリアルについては、[Using Fortanix Data Security Manager with EDB Postgres for TDE](#) を参照してください。

Entrust KeyControl 統合ガイド

インストール、構成、および使用手順キーのローテーションを含むについては、[EDB Postgres and Entrust KeyControl 統合ガイド](#)を参照してください。

Disabling the key wrapping

テストなどでキーラッピングをしたくない場合は、wrap および unwrap コマンドを特別な値- に設定する必要があります。

この設定は、それ以上の処理をせずにファイルのキーを使用することを指定します。このアプローチは、wrap または unwrap コマンドをまったく設定しないこと、およびいずれか/両方を空の文字列に設定することとは異なります。透過的データ暗号化が使用されるときに wrap または unwrap コマンドが設定されていないと、データ暗号化キーが安全でないままで、影響を受けるユーティリティプログラムを実行すると致命的エラーが発生します。

Rotating the data encryption key

マスターキーを変更するには、古いキーを指定して unwrap コマンドを手動で実行します。次に、結果を wrap コマンドにフィードして、新しいキーを指定します。同様に、データキーがパスフレーズで保護されている場合、パスフレーズを変更するには、古いパスフレーズを使用して unwrap コマンドを実行します。次に、新しいパスフレーズを使用して結果を wrap コマンドにフィードします。データベースサーバーの実行中にこれらの操作を実行できます。ファイル内のラップされたデータキーは、起動時にのみ使用されます。サーバーの実行中は使用されません。

パスフレーズのローテーション

openssl を使用する [Using a passphrase](#) の例を基にパスフレーズを変更すると、次のことができます。

```
cd $PGDATA/pg_encryption/  
openssl enc -d -aes-128-cbc -pbkdf2 -in key.bin | openssl enc -e -aes-128-cbc -pbkdf2 -  
→-out key.bin.new  
mv key.bin.new key.bin
```

この方法では、復号化と暗号化コマンドが端末でパスフレーズを同時に要求しますが、これは厄介で混乱を招きます。代替案は次のとおりです。

```
cd $PGDATA/pg_encryption/
openssl enc -d -aes-128-cbc -pbkdf2 -in key.bin -pass pass:<replaceable>
→ACTUALPASSPHRASE</replaceable> | openssl enc -e -aes-128-cbc -pbkdf2 -out key.bin.
→new
mv key.bin.new key.bin
```

この手法は、古いパスフレーズを漏洩させますが、いずれにせよ置き換えられます。openssl は、パスフレーズを提供する他の多くの方法をサポートしています。

キーストアのラッピングキーのローテーション

Using a key store を使用する場合、たとえば、unwrap および wrap コマンドを同様に接続できます。

```
cd $PGDATA/pg_encryption/
crypt decrypt aws --in key.bin --region us-east-1 | crypt encrypt aws --out key.bin.
→new --region us-east-1 --kms alias/pg-tde-master-2
mv key.bin.new key.bin
```

注釈: 既存のデータディレクトリのデータキーマスターキーによってラップされたキーは変更できません。それを行う必要がある場合は、pg_dump、pg_upgrade、または論理レプリケーションを使用したアップグレードプロセスでデータディレクトリを実行する必要があります。

2.3 Using TDE initialization options

TDE 対応サーバーの初期化には、2つの必須設定が必要です。1つはTDEを有効にし、もう1つはデータ暗号化キーを保護します。

2.3.1 TDE を有効にするには

TDE 対応データベースサーバーを作成するには、--data-encryption を使用する必要があります。これは、サーバーを暗号化するデータ暗号化キーを作成します。

pg_upgrade のターゲットとして新しいクラスターを準備するときに既存のクラスターからキーをコピーする場合は、オプション --copy-key-from= <オプション --copy-key-from=<file>>` を追加して使用します。

2.3.2 データ暗号化キーを保護するには

TDE 対応データベースを作成するとき、TDE はユーザーに対して透過的なデータ暗号化キーを生成します。

このキーをラップするには、フォームまたはラッピングおよびラップ解除コマンドの追加の保護メカニズムが必要であり、データベースサーバーで利用できるようにする必要があります。

使用可能な保護メカニズムの概要、およびこの構成を `initdb` に提供する方法の例については、[TDE にラッピングおよびアンラップコマンドを提供する](#) を参照してください。

2.3.3 オプションリファレンス

すべての必須および選択オプション、およびサポートされている値の概要については、[initdb TDE options](#) を参照してください。

2.4 Working with encrypted files

特定の Postgres ユーティリティと操作では、ファイルにアクセスしてデータを読み取り、必要な出力を配信する必要があります。

TDE 対応データベースサーバーでは、データが暗号化されるため、復号メカニズムがないとアクセスできません。これらのユーティリティが暗号化されたファイルで読み取り操作を実行できるようにするには、復号メカニズムを提供する必要があります。

- [Working with encrypted backup files](#) は、バックアップファイルに関する問題の作業およびトラブルシューティングの方法に関するガイダンスを提供します。
- [Working with encrypted WAL files](#) は、WAL ファイルの問題の作業とトラブルシューティングの方法に関するガイダンスを提供します。

2.4.1 Working with encrypted WAL files

TDE が有効になっている場合、WAL ファイルは暗号化されます。暗号化された WAL ファイルで操作を実行する場合は、操作でファイルを復号化することを許可する必要があります。

暗号化された WAL ファイルを使用してトラブルシューティングを行う場合、WAL コマンドオプションを使用できます。

TDE 暗号化された WAL ファイルのダンプ

暗号化された WAL ファイルを操作するには、`pg_waldump` ユーティリティが `unwrap` キーにアクセスし、復号化できることを確認する必要があります。この目的のために、ユーティリティは3つの値の意識を必要とします。

次のオプションを使用してこれらの値を `pg_waldump` コマンドに渡します。TDE 対応クラスターを初期化するときには使用したのと同じ値を使用してください。

`-data-encryption`

WAL ファイルが透過的データ暗号化で暗号化された場合に、このオプションを指定します。

`--data-encryption` または `-y` オプションは、コマンドが暗号化を認識できるようにします。それ以外の場合、`pg_waldump` は、WAL ファイルが暗号化されているかどうかを検出できません。

TDE 対応データベースクラスターを初期化するときには使用したのと同じ暗号化構成を提供します。たとえば、クラスターの作成中に AES キー長を指定した場合、ここでも同様に指定する必要があります。それ以外の場合、値を指定せずにフラグを実行します。詳細については、[initdb TDE options](#) を参照してください。

`-key-file-name=`

`--key-file-name=<file>` オプションを使用して、WAL ファイルの復号化に必要なデータ暗号化キーを含むファイルを参照します。`pg_encryption/key.bin` ファイルの場所を提供します。このファイルは、暗号化を有効にしてクラスターを初期化するときに生成されます。

コマンドは、指定された場所からデータ暗号化キーをロードできます。

`-key-unwrap-command=`

`--key-unwrap-command=<command>` オプションの場合、TDE クラスターの初期化時にデータ暗号化キーをアンラップ (復号) するように指定した `decryption` コマンドを提供します。詳細については、[initdb TDE options](#) を参照してください。

または、`pg_waldump` コマンドを実行する前に `PGDATAKEYUNWRAPCMD` 環境変数を設定できます。`--key-unwrap-command=<command>` オプションが指定されていない場合、`pg_waldump` は `PGDATAKEYUNWRAPCMD` にフォールバックします。この [Creating a database with TDE](#) は、環境変数をエクスポートする方法を示しています。

例

この例では、`pg_walddump` を使用して、`openssl` を使用してデータ暗号化キーをラップする暗号化されたクラスタの WAL ログを表示します。

```
pg_walddump --data-encryption --key-file-name=pg_encryption/key.bin --key-unwrap-  
→command=openssl enc -d -aes-128-cbc -pass pass:<passphrase> -in %p
```

破損した TDE 暗号化された WAL ファイルのリセット

破損した暗号化された WAL ファイルをリセットするには、`pg_resetwal` コマンドが `unwrap` キーにアクセスし、復号化できることを確認する必要があります。次のオプションを使用して `unwrap` コマンドのキーを `pg_resetwal` コマンドに渡すか、フォールバック環境変数に依存することができます。

`--key-unwrap-command=`

`--key-unwrap-command=<command>` オプションの場合、TDE クラスタの初期化時にデータ暗号化キーをアンラップ (復号) するように指定した decryption コマンドを提供します。詳細については、*initdb TDE options* を参照してください。

または、`pg_resetwal` コマンドを実行する前に `PGDATAKEYUNWRAPCMD` 環境変数を設定できます。`--key-unwrap-command=<command>` オプションが指定されていない場合、`pg_resetwal` は `PGDATAKEYUNWRAPCMD` にフォールバックします。この *Creating a database with TDE* は、環境変数をエクスポートする方法を示しています。

例

この例では、`pg_resetwal` を使用して、`openssl` を使用してデータ暗号化キーをラップする暗号化クラスタの破損した暗号化された WAL ログをリセットします。

```
pg_resetwal --key-unwrap-command=openssl enc -d -aes-128-cbc -pass pass:<passphrase> -  
→in %p
```

2.4.2 Working with encrypted backup files

TDE が有効になっている場合、バックアップファイルは暗号化されます。暗号化されたバックアップファイルに対して操作を実行する場合は、操作でファイルを復号化することを許可する必要があります。

TDE システムのバックアップを検証する

暗号化されたバックアップファイルを検証するには、`pg_verifybackup` コマンドが `unwrap` キーを認識する必要があります。次のオプションを使用して `unwrap` コマンドのキーを `pg_verifybackup` コマンドに渡すか、フォールバック環境変数に依存することができます。

`--key-unwrap-command=<command>`

データ暗号化キーをアンラップ (復号) するコマンドを指定します。コマンドには、ラップされたキーを読み取るファイルを指定するプレースホルダー `%p` を含める必要があります。コマンドは、ラップされていないキーを標準出力に書き込む必要があります。このオプションを指定しない場合、環境変数 `PGDATAKEYUNWRAPCMD` が使用されます。キーのラップ解除コマンドを適用したくない場合は、特別な値 `-` を使用します。データ暗号化を使用している場合、このオプションまたは環境変数 `フォールバック` を指定する必要があります。詳細については、[Securing the data encryption key](#) を参照してください。

TDE システムのタイムラインを再同期する

暗号化されたクラスターとそのバックアップを再同期するには、`pg_rewind` コマンドが `unwrap` キーを認識する必要があります。次のオプションを使用して `unwrap` コマンドのキーを `pg_rewind` コマンドに渡すか、フォールバック環境変数に依存することができます。

`-key-unwrap-command=`

データ暗号化キーをアンラップ (復号) するコマンドを指定します。コマンドには、ラップされたキーを読み取るファイルを指定するプレースホルダー `%p` を含める必要があります。コマンドは、ラップされていないキーを標準出力に書き込む必要があります。このオプションを指定しない場合、環境変数 `PGDATAKEYUNWRAPCMD` が使用されます。

キーのラップ解除コマンドを適用したくない場合は、特別な値 `-` を使用します。データ暗号化を使用している場合、このオプションまたは環境変数 `フォールバック` を指定する必要があります。詳細については、[Securing the data encryption key](#) を参照してください。

2.5 Upgrading a TDE-enabled database

`pg_upgrade` を追加の TDE 引数とともに使用して、多数のアップグレードおよび移行操作を実行できます。この表は、サポートされているユースケースの概要を提供します。

Use case	Source unencrypted server	Target encrypted server
Perform a minor upgrade and add encryption	Unencrypted EDB Postgres Extended Server 16.1	Encrypted EDB Postgres Extended Server 16.2

次のページに続く

表 1 – 前のページからの続き

Use case	Source unencrypted server	Target encrypted server
Change the Postgres distribution and add encryption	Unencrypted PostgreSQL 16	Encrypted EDB Postgres Advanced Server 16
Maintain the Postgres distribution and add encryption	Unencrypted EDB Postgres Advanced Server 15	Encrypted EDB Postgres Advanced Server 15
Maintain the Postgres distribution and rotate encryption keys	Encrypted EDB Postgres Advanced Server 15	Encrypted EDB Postgres Advanced Server 15 with new encryption keys

重要: ソースサーバーとターゲットサーバーは両方とも同じ Postgres メジャーバージョンである必要があります。 `pg_upgrade` は、マイナーバージョン間のアップグレードのみをサポートしています。

2.5.1 チュートリアル

さまざまな移行シナリオに焦点を当てたステップバイステップチュートリアルのリストについては、[Tutorials](#) を参照してください。

2.6 Tutorials

`initdb` を使用して TDE 対応データベースサーバーを作成します。

または、`initdb` で TDE 対応データベースサーバーを作成し、`pg_upgrade` でデータを移行することにより、既存のデータベースインスタンスを移行します。

2.6.1 Creating a database with TDE

TDE を有効にして新しい EDB Postgres Advanced Server クラスターを作成します。

- 暗号化のための `wrap` および `unwrap` コマンドをエクスポートするように環境変数を設定します。
- 暗号化を有効にしてサーバーを初期化します。
- データベースサーバーを起動します。
- TDE が有効になっていることを確認します。

実際の例

この例では、Linux プラットフォームで実行されている EDB Postgres Advanced Server 15 を使用します。openssl を使用してパスフレーズを定義し、生成されたデータ暗号化キーをラップおよびラップ解除します。

1. データ暗号化キー wrap と復号化 unwrap 環境変数を設定します。

```
export PGDATAKEYWRAPCMD=openssl enc -e -aes-128-cbc -pass pass:ok -out %p
export PGDATAKEYUNWRAPCMD=openssl enc -d -aes-128-cbc -pass pass:ok -in %p
```

注釈:

- Windows を使用している場合、変数値を一重引用符で囲む必要はありません。-ok を、データ暗号化キーのラップに使用するパスフレーズに置き換えることを確認します。!!!

1. 暗号化を有効にした initdb を使用してクラスターを初期化します。このコマンドは、postgresql.conf ファイルの data_encryption_key_unwrap_command パラメーターを設定します。

```
/usr/edb/as15/bin/initdb --data-encryption -D /var/lib/edb/as15/data
```

1. クラスターを起動します。

```
/usr/edb/as15/bin/pg_ctl -D /var/lib/edb/as15/data start
```

1. postgresql.conf で grep を実行して、data_encryption_key_unwrap_command の設定を確認します。

```
grep data_encryption_key_unwrap_command /var/lib/edb/as15/data/postgresql.conf
__OUTPUT__
data_encryption_key_unwrap_command = openssl enc -d -aes-128-cbc -pass pass:ok -in %p
```

1. *Verifying TDE is enabled*。

2.6.2 Enabling TDE on an existing EDB Postgres Advanced Server

TDE を有効にして新しい EDB Postgres Advanced Server クラスターを作成し、pg_upgrade を使用して既存のソースクラスターから新しい暗号化されたクラスターにデータを転送します。

- **アップグレードの準備** 既存のインスタンスのバックアップを実行する。
- **暗号化されたサーバーの作成**
 - 新しいサーバーの空のディレクトリを作成し、enterprisedb がそれを所有していることを確認します。
 - 暗号化のために wrap および unwrap コマンドをエクスポートするように環境変数を設定します。
 - 暗号化を有効にしてサーバーを初期化します。
 - デフォルトポートを変更して、新しいサーバーが別のポートで利用できるようにします。

- データベースサーバーを起動します。
- データベースサーバーに接続し、機能していることを確認します。

- **暗号化されたサーバーへのアップグレード**

- ソースと新しいサーバーの両方を停止します。
- `pg_upgrade` を `--copy-by-block` オプションとともに使用して、ソースサーバーから新しいサーバーにデータをコピーします。ソース、ターゲットの `bin`、およびデータディレクトリを指定します。
- 新しい暗号化されたデータベースサーバーを起動します。
- 暗号化されたデータベースサーバーに接続し、データが転送されたことを確認します。

- **アップグレード後のクリーンアップ**

- データベースとその統計をクリーンアップします。
- `pg_upgrade` が提供するスクリプトを使用して、ソース EDB Postgres Advanced Server クラスターを削除します。

実際の例

この例では、Ubuntu 22.04 マシンで実行されている EDB Postgres Advanced Server バージョン 16 で透過的データの暗号化を有効にします。

同様のワークフローは、EDB Postgres Advanced Server および EDB Postgres Extended Server の他のバージョンに適用されます。BIN および CONFIG ディレクトリの場所は、オペレーティングシステムと Postgres バージョンによって異なることに注意してください。

アップグレードの準備

`pg_dumpall`、`pgBackRest`、または `Barman` を使用して、暗号化されていないソースサーバーのバックアップを作成します。

暗号化されたサーバーの作成

1. 新しいサーバーの空のディレクトリを作成します。この例では、ディレクトリ名は TDE です。

```
mkdir /var/lib/edb-as/16/TDE
```

1. `enterprisedb` ユーザーがディレクトリを所有していることを確認します。

```
sudo chown enterprisedb /var/lib/edb-as/16/TDE
sudo chgrp /var/lib/edb-as/16/TDE
```

1. `wrap` および `unwrap` コマンドをエクスポートするように環境変数を設定します。

```
export PGDATAKEYWRAPCMD=openssl enc -e -aes-128-cbc -pass pass:ok -out %p
export PGDATAKEYUNWRAPCMD=openssl enc -d -aes-128-cbc -pass pass:ok -in %p
```

注釈: または、暗号化されたサーバーを初期化するときに `--key-unwrap-command=<command>` および `--key-wrap-command=<command>` 引数を使用して、`wrap` および `unwrap` コマンドを含めます。考えられる構成の詳細については、[initdb TDE options](#) を参照してください。

1. 暗号化を使用して新しいサーバーを初期化します。

```
/usr/lib/edb-as/16/bin/initdb --data-encryption -D /var/lib/edb-as/16/TDE
```

このコマンドは、暗号化されたサーバーのすべての構成ファイルで CONFIG ディレクトリを初期化します。

1. 暗号化されたインスタンスの構成ファイルのポート番号を変更します。 `#port` という行のコメントを外し、ポート番号を変更します。例

```
port                                5590
```

1. 暗号化されたサーバーを起動します。

```
/usr/lib/edb-as/16/bin/pg_ctl -D /var/lib/edb-as/16/TDE -l logfile start
```

1. サーバーに接続します。

```
/usr/lib/edb-as/16/bin/psql -p 5590 edb
```

注釈: 2つの異なる Postgres バージョンを使用している場合、暗号化されたサーバーの `psql` ユーティリティを使用します。それ以外の場合、システムは以前のインスタンスから `psql` を使用しようとします。

1. 新しいサーバーが暗号化されるように保証には、[Verifying TDE is enabled](#)。

暗号化されたサーバーへのアップグレード

1. 両方のサーバーを停止します。

```
/usr/lib/edb-as/16/bin/pg_ctl -D /var/lib/edb-as/16/non-TDE stop
/usr/lib/edb-as/16/bin/pg_ctl -D /var/lib/edb-as/16/TDE stop
```

1. 非互換性をテストするには、`check` モードで `pg_upgrade` コマンドを実行します。 `-b` および `-B` では、ソースおよびターゲット BIN ディレクトリを指定します。 `-d` および `-D` では、ソースおよびターゲットの CONFIG ディレクトリを指定します。 `--copy-by-block` オプションを含めます。

```
/usr/lib/edb-as/16/bin/pg_upgrade -b /usr/lib/edb-as/16/bin -B /usr/lib/edb-as/16/bin
→ -d /var/lib/edb-as/16/non-TDE -D /var/lib/edb-as/16/TDE --copy-by-block --check
```

注釈: `--check` モードは、コマンドを実行せずに事前チェックを実行します。

1. ソースサーバーからターゲットサーバーにデータをコピーするには、通常モードで `pg_upgrade` コマンドを実行します。

```
/usr/lib/edb-as/16/bin/pg_upgrade -b /usr/lib/edb-as/16/bin -B /usr/lib/edb-as/16/bin  
→ -d /var/lib/edb-as/16/non-TDE -D /var/lib/edb-as/16/TDE --copy-by-block
```

1. 暗号化されたサーバーを再起動します。

```
/usr/lib/edb-as/16/bin/pg_ctl -D /var/lib/edb-as/16/TDE start
```

1. 暗号化されたデータベースサーバーに接続します。

```
/usr/lib/edb-as/16/bin/psql -p 5590 edb
```

1. スポットチェックを実行して、暗号化されていないサーバーにあったデータベース、テーブル、スキーマ、およびリソースが新しいサーバーで利用できることを確認します。例、すべてのデータベースをリストします。

```
\l
```

データベースたとえば `hr` に接続し、既存のテーブルを検索します。

```
\c hr  
SELECT * FROM dept;
```

アップグレード後のクリーンアップ

`pg_upgrade` がデータを正常に暗号化したことを確認した後、クリーンアップを実行します。

1. データベースとその統計をクリーンアップします。

```
/usr/lib/edb-as/16/bin/vacuumdb --all --analyze-in-stages
```

1. `pg_upgrade` によって生成されたスクリプトを使用して、暗号化されていないサーバーのすべてのデータファイルを削除します。

```
./delete_old_cluster.sh
```

2.6.3 Upgrading PostgreSQL to EDB Postgres Extended Server while enabling TDE

TDE を有効にして新しい EDB Postgres Extended Server クラスターを作成し、`pg_upgrade` を使用して既存の PostgreSQL クラスターから新しい暗号化されたクラスターにデータを転送します。

- **アップグレードの準備** 既存のインスタンスのバックアップを実行する。
- **暗号化されたサーバーの作成**.
 - 新しいサーバーの空のディレクトリを作成し、postgres ユーザーがそれを所有していることを確認します。
 - 暗号化のために `wrap` および `unwrap` コマンドをエクスポートするように環境変数を設定します。
 - 暗号化を有効にしてサーバーを初期化します。
 - デフォルトポートを変更して、新しいサーバーが別のポートで利用できるようにします。
 - データベースサーバーを起動します。
 - データベースサーバーに接続し、機能していることを確認します。
- **暗号化されたサーバーへのアップグレード**.
 - ソースと新しいサーバーの両方を停止します。
 - `pg_upgrade` を `--copy-by-block` オプションとともに使用して、ソースサーバーから新しいサーバーにデータをコピーします。ソース、ターゲットの `bin`、およびデータディレクトリを指定します。
 - 新しい暗号化されたデータベースサーバーを開始します。
 - 暗号化されたデータベースサーバーに接続し、データが転送されたことを確認します。
- **アップグレード後のクリーンアップ**.
 - データベースとその統計をクリーンアップします。
 - `pg_upgrade` が提供するスクリプトを使用して、ソース PostgreSQL クラスターを削除します。

実際の例

この例では、Ubuntu 22.04 マシンで透過的データの暗号化を有効にしながら、PostgreSQL 16 インスタンスを EDB Postgres Extended Server 16 にアップグレードします。bin および config ディレクトリの場所は、オペレーティングシステムと Postgres バージョンによって異なります。

アップグレードの準備

- [EDB repository](#) から EDB Postgres Extended Server をインストールします。インストールするバージョンのメジャーバージョンがソースサーバーと同じであることを確認します。pg_upgrade は、マイナーバージョンとパッチバージョン間のアップグレードをサポートしていますが、異なるメジャーバージョン間のアップグレードはサポートしていません。
- pg_dumpall、pgBackRest、または Barman を使用して、暗号化されていないソースサーバーのバックアップを作成します。

暗号化されたサーバーの作成

1. 新しいサーバーの空のディレクトリを作成します。この例では、ディレクトリ名は TDE です。

```
mkdir /var/lib/edb-pge/16/TDE
```

1. postgres ユーザーがディレクトリを所有していることを確認します。

```
sudo chown postgres /var/lib/edb-pge/16/TDE
sudo chgrp /var/lib/edb-pge/16/TDE
```

1. wrap および unwrap コマンドをエクスポートするように環境変数を設定します。

```
export PGDATAKEYWRAPCMD=openssl enc -e -aes-128-cbc -pass pass:ok -out %p
export PGDATAKEYUNWRAPCMD=openssl enc -d -aes-128-cbc -pass pass:ok -in %p
```

注釈: または、暗号化されたサーバーを初期化するとき `--key-unwrap-command=<command>` および `--key-wrap-command=<command>` 引数を使用して、wrap および unwrap コマンドを含めます。考えられる構成の詳細については、[initdb TDE options](#) を参照してください。

1. 暗号化を使用して新しいサーバーを初期化します。

```
/usr/lib/edb-pge/16/bin/initdb --data-encryption -D /var/lib/edb-pge/16/TDE
```

このコマンドは、暗号化されたサーバーのすべての構成ファイルで config ディレクトリを初期化します。

1. 暗号化されたインスタンスの構成ファイルのデフォルトのポート番号を変更します。#port という行のコメントを外し、ポート番号を変更します。例

```
port                    5590
```

1. 暗号化されたサーバーを起動します。

```
/usr/lib/edb-pge/16/bin/pg_ctl -D /var/lib/edb-pge/16/TDE start
```

1. サーバーに接続します。

```
/usr/lib/edb-pge/16/bin/psql -p 5590
```

注釈: 2つの異なる Postgres バージョンを使用している場合、暗号化されたサーバーの psql ユーティリティを使用します。それ以外の場合、システムは以前のインスタンスから psql を使用しようとします。

1. 新しいサーバーが暗号化されるように保証には、*Verifying TDE is enabled*。

暗号化されたサーバーへのアップグレード

1. 両方のサーバーを停止します。

```
/usr/lib/postgresql/16/bin/pg_ctl -D /var/lib/postgresql/16/non-TDE stop
/usr/lib/edb-pge/16/bin/pg_ctl -D /var/lib/edb-pge/16/TDE stop
```

1. 非互換性をテストするには、check モードで pg_upgrade コマンドを実行します。-b および-B では、ソースおよびターゲット BIN ディレクトリを指定します。-d および-D では、ソースおよびターゲットの config ディレクトリを指定します。--copy-by-block オプションを含めます。

```
/usr/lib/edb-pge/16/bin/pg_upgrade -b /usr/lib/postgresql/16/bin -B /usr/lib/edb-pge/
→16/bin -d /var/lib/postgresql/16/non-TDE -D /var/lib/edb-pge/16/TDE --copy-by-block
→--check
```

注釈: --check モードは、コマンドを実行せずに事前チェックを実行します。

1. ソースサーバーからターゲットサーバーにデータをコピーするには、通常モードで pg_upgrade コマンドを実行します。

```
/usr/lib/edb-pge/16/bin/pg_upgrade -b /usr/lib/postgresql/16/bin -B /usr/lib/edb-pge/
→16/bin -d /var/lib/postgresql/16/non-TDE -D /var/lib/edb-pge/16/TDE --copy-by-block
```

1. 暗号化されたサーバーを再起動します。

```
/usr/lib/edb-pge/16/bin/pg_ctl -D /var/lib/edb-pge/16/TDE start
```

1. 暗号化されたデータベースサーバーに接続します。

```
/usr/lib/edb-pge/16/bin/psql -p 5590
```

1. スポットチェックを実行して、暗号化されていないサーバーにあったデータベース、テーブル、スキーマ、およびリソースが新しいサーバーで利用できることを確認します。例、すべてのデータベースをリストします。

```
\l
```

データベースたとえば **hr** に接続し、既存のテーブルを検索します。

```
\c hr
SELECT * FROM dept;
```

アップグレード後のクリーンアップ

`pg_upgrade` がデータを正常に暗号化したことを確認した後、クリーンアップを実行します。

1. postgres ユーザーとして、データベースとその統計をクリーンアップします。

```
/usr/lib/edb-pge/16/bin/vacuumdb --all --analyze-in-stages
```

1. `pg_upgrade` によって生成されたスクリプトを使用して、暗号化されていないサーバーのすべてのデータファイルを削除します。

```
./delete_old_cluster.sh
```

2.6.4 Verifying TDE is enabled

`pg_control_init` テーブルの `data_encryption_version` 列を照会することにより、サーバーに TDE が存在するかどうかを知ることができます。

値 0 は、TDE が有効になっていないことを意味します。ゼロ以外の値は、使用中の TDE のバージョンを反映しています。現在、TDE が有効になっている場合、この値は 1 です。

```
select data_encryption_version from pg_control_init();
__OUTPUT__
data_encryption_version
-----
1
(1 row)
```

第3章 Reference

3.1 Commands affected by TDE

TDE が有効になっている場合、次のコマンドには TDE 固有のオプションがあるか、環境変数または構成ファイルの TDE 設定を読み取ります。

- *initdb TDE options*
- *pg_waldump*
- *pg_resetwal*
- *pg_verifybackup*
- *pg_rewind*
- *pg_upgrade TDE options*

3.2 initdb TDE options

暗号化を有効にするには、`initdb` コマンドで次のオプションを使用します。

3.2.1 オプション `--data-encryption` または `-y`

データベースサーバーの初期化時に、透過的なデータの暗号化を追加します。

サポートされている値

オプションで AES キー長を指定できます。有効な値は 128 および 256 です。デフォルトは 128 です。

3.2.2 オプション `--key-wrap-command=<command>`

データ暗号化キーを保護するためのラッピング/暗号化コマンドを提供します。

サポートされている値

`<command>` はカスタマイズ可能ですが、プレースホルダー `%p` を含める必要があります。例と情報については、*TDE にラッピングおよびアンラップコマンドを提供する* を参照してください。使用可能なラッピングメカニズムの概要については、*Securing the data encryption key* を参照してください。

指定しない場合、TDE は環境変数 `PGDATAKEYWRAPCMD` にフォールバックします。

ラッピングメカニズムを適用したくない場合は、`-` を使用します。

3.2.3 オプション `--key-unwrap-command=<command>`

データ暗号化キーにアクセスするための解凍/復号コマンドを提供します。

サポートされている値

`<command>` はカスタマイズ可能ですが、プレースホルダー `%p` を含める必要があります。例と情報については、*Configuring wrapping commands* を参照してください。

指定しない場合、TDE は環境変数 `PGDATAKEYUNWRAPCMD` にフォールバックします。

ラッピングメカニズムを適用しなかった場合は、`-` を使用します。

3.2.4 オプション `--no-key-wrap`

キーのラッピングを無効にします。データ暗号化キーは、代わりにデータディレクトリに平文で保存されます。このオプションは、`wrap` コマンドと `unwrap` コマンドの両方を特別な値 `-` に設定するためのショートカットです。

注釈: このオプションの使用は安全ではありません。テスト目的でのみ使用してください。!!!データの暗号化を選択し、このオプションを指定しない場合、`key Wrap` および `unwrap` コマンドを提供する必要があります。それ以外の場合、`initdb` はエラーで終了します。

サポートされている値

`--no-key-wrap` オプションは、値を指定する必要はありません。

3.2.5 オプション `--copy-key-from=<file>`

指定された場所から既存のデータ暗号化キーをコピーします。たとえば、`pg_upgrade` でのアップグレード中に既存のサーバーからキーを再利用する場合。

サポートされている値

`<file>` は、既存のキーのディレクトリです。通常、暗号化キーは `pg_encryption/key.bin` に保存されます。

3.3 `pg_upgrade` TDE options

`pg_upgrade` へのこれらの引数は、暗号化されたクラスターのアップグレードに役立ちます。

3.3.1 オプション `--copy-by-block`

デフォルトのファイル全体を一度にコピーするのではなく、ブロックごとに新しいクラスターにファイルをコピーします。このオプションはデフォルトモードと同じですが、速度が遅くなります。ただし、暗号化設定が異なるクラスター間のアップグレードはサポートしています。異なる暗号化設定を使用するクラスター間でアップグレードする場合、つまり、非暗号化から暗号化、暗号化から非暗号化、またはその両方が異なるキーで暗号化されているクラスター間でアップグレードする場合、このオプションを使用する必要があります。新しいクラスターにファイルをコピーしているときに、新しいクラスターのキーと設定を使用してファイルを復号し、再暗号化します。古いクラスターが暗号化され、新しいクラスターが非暗号化として初期化された場合、このオプションは古いクラスターからデータを復号し、暗号化せずに新しいクラスターにコピーします。古いクラスターが暗号化されておらず、新しいクラスターが暗号化されたものとして初期化された場合、このオプションは古いクラスターからデータを暗号化し、暗号化された新しいクラスターに配置します。`pg_upgrade` のターゲットとして新しいクラスターを準備するときに、既存のクラスターからキーをコピーする方法については、[initdb TDE options](#) の説明を参照してください。 `--copy-by-block` の使用例については、[Tutorials](#) を参照してください。

3.3.2 オプション `--key-unwrap-command=<command>`

データ暗号化キーのラップを解除し復号化し、コピーするファイルにアクセスするコマンドを指定します。これは、サーバーの初期化中に指定した `unwrap` コマンドと同じである必要があります。

このオプションを指定しない場合、`pg_upgrade` は環境変数 `PGDATAKEYUNWRAPCMD` を読み取ります。

3.4 Limitations

3.4.1 FILE_COPY

透過的データ暗号化が有効になっている場合、`CREATE DATABASE` で `strategy` パラメーターで `FILE_COPY` ストラテジーを使用することはできません。

詳細については、[PostgreSQL CREATE DATABASE documentation](#) を参照してください。