Learning Objectives

Learners will be able to...

- describe how a secure TCP/IP connection with SSL configured
- list alternative methods of secure connections
- describe transparent data encryption
- use pgcrypto module for per-user encryption

info

Make Sure You Know

Familiarize yourself with the concept of secure connections, including the use of encryption protocols.

Have a basic understanding of PostgreSQL, as pgcrypto is a module used for cryptographic functions in PostgreSQL.

Familiarize yourself with common cryptographic functions provided by pgcrypto, such as encryption and decryption.

SSL connection between client and server

PostgreSQL has built-in support for encrypting connections between the user and the server to prevent man-in-the-middle attacks.

> info
>
> ## SSL vs TLS
>
> The terms SSL and TLS are often used interchangeably to mean a secure encrypted connection using a TLS protocol. SSL protocols are the precursors to TLS protocols, and the term SSL is still used for encrypted connections even though SSL protocols are no longer supported.

The group of settings with the prefix `ssl` is responsible for setting up this encryption.

In the current installation they are in the file `/etc/postgresql/14/main/postgresql.conf` from line `105`. You can view it by clicking on the button.

Option	Description
`ssl`	**on** or **off** - Enables SSL connections
`ssl_cert_file`	Specifies the name of the file containing the SSL server certificate
`ssl_key_file`	Specifies the name of the file containing the SSL server private key.

In this case, the setting is enabled and standard certificates from the PostgreSQL package are used as certificates(`/etc/ssl/certs/ssl-cert-snakeoil.pem` and `/etc/ssl/private/ssl-cert-snakeoil.key`).

Older installations may not have encryption enabled by default. To activate it, you need to create either a self-written certificate or order a certificate from a certificate provider. After that, add files to the system and restart the service.

To ensure that only an encrypted connection is used, you can use the `hostssl` setting described in the previous topic.

Other secure connection types

TLS is not the only way to establish a secure connection between a client and a server.

GSSAPI Encryption

PostgreSQL also has native support for using GSSAPI to encrypt client/server communications for increased security.

The PostgreSQL server will listen for both normal and GSSAPI-encrypted connections on the same TCP port, and will negotiate with any connecting client whether to use GSSAPI for encryption.

You can force the client to use this connection type with a `gss` setting in `pg_hba.conf` file.

SSH Tunnels

It is possible to use SSH to encrypt the network connection between clients and a PostgreSQL server.

To do this, you need to create a SSH tunnel between the remote machine and yours.

```
ssh -L 63333:localhost:5432 you.database.domain
```

The first number in the -L argument, 63333, is the local port number on your machine. The second number, 5432, the port number your database server is using.

After that you will be able to connect to the database using the local port.

```
psql -h localhost -p 63333 postgres
```

Database encryption solution

Database data protection with encryption can be implemented at several levels:

- OS level discs encryption
- Transparent data encryption on database side
- User level data encryption

Hard drive encryption

Encrypting the system disk is convenient because in this case no changes to the database configuration are required at all. This also makes additional metadata (clog and textual log) encrypted.

Different utilities and encryption solutions may be used for different operating systems.

- **Linux** - The easiest way is to set up disk encryption during system installation. For this, LUKS technology is used. Or use one of the many 3d party utilities.
- **macOS** - You can encrypt a drive in macOS using the standard FileVault tool.
- **Windows** - There are two popular ways to encrypt a drive in Windows:
  - using VeraCrypt;
  - using standard BitLocker encryption.

Transparent data encryption

Transparent Data Encryption (often abbreviated to TDE) is a technology employed by Microsoft, IBM and Oracle to encrypt database files. TDE offers encryption at file level.

This type of encryption is used when the database administrator cannot or does not rely on the file system for confidentiality.

Unfortunately, PostgreSQL does not support this technology, but there are solutions from companies that allow you to do this. For example, CYBERTEC offers an Transparent Data Encryption patch for PostgreSQL.

The community is also actively arguing whether this solution should be implemented or whether disk encryption at the OS level is enough.

Per-user encryption

Pgcrypto can be used to encrypt user data instead of a solution that will encrypt everything.

First you need to activate this extension.

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

To encrypt any string with a symmetric encryption method, we can use the function `PGP_SYM_ENCRYPT`. The first parameter of this function is the data, the second is the encryption key.

```
SELECT PGP_SYM_ENCRYPT('super secret data', 'secretkey')::text;
```

For decryption, the `PGP_SYM_DECRYPT` function is used.

```
SELECT PGP_SYM_DECRYPT(

        '\xc30d040703020395313d44278d766fd242014fa939641f999913d291ba79a7532e582f9934392764d3639eacf1
        3467e0b35ab2d8591acd9521d75',


  'secretkey'
);
```

You could store encryption results directly into table. Create a structure.

```
CREATE TABLE user_secret (
  id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  data TEXT
);
```

And insert data:

```
INSERT INTO user_secret (data)
VALUES (
  PGP_SYM_ENCRYPT('my super secret data', 'secretkey')
);
```

Instead of `secretkey` you could use even own string.

Now you can get the encrypted data from the table.

```
SELECT
  id,
  PGP_SYM_DECRYPT(data::bytea, 'secretkey') as data
FROM user_secret WHERE id = 1;
```

If you used your own key, don't forget to change the default one!

Otherwise, you will see the following error.

```
ERROR:  Wrong key or corrupt data
```

Also this module provides the following features:
- General Hashing Functions, like `md5`, `sha1`, `sha256`
- Password Hashing Functions, like Blowfish or Extended DES
- Public-key encryption
- Random-Data Functions