django-pgcrypto-expressions Documentation

Release 0.1

Carl Meyer

January 11, 2016

Contents

pgcrypto for Django models.

CHAPTER 1

Prerequisites

`django-pgcrypto-expressions` supports Django 1.8.2 and later on Python 2.7, 3.4, pypy, and pypy3. PostgreSQL is required.

CHAPTER 2

Installation

`django-pgcrypto-expressions` is available on PyPI. Install it with:

```
pip install django-pgcrypto-expressions
```

CHAPTER 3

Setup

Your database must have the pgcrypto extension installed. You can install it by running `CREATE EXTENSION pgcrypto;`.

CHAPTER 4

Usage

Just import and use the included field classes in your models:

```
from django.db import models
from pgcrypto_expressions.fields import EncryptedTextField


class MyModel(models.Model):
    name = EncryptedTextField()
```

You can assign values to and read values from the `name` field as usual, but the values will automatically be encrypted using pgcrypto's `pgp_sym_encrypt` function when you save it, and decrypted using `pgp_sym_decrypt` when you load it from the database.

4.1 Field types

Several other field classes are included: `EncryptedCharField`, `EncryptedEmailField`, `EncryptedIntegerField`, `EncryptedDateField`, and `EncryptedDateTimeField`. All field classes accept the same arguments as their non-encrypted versions.

To create an encrypted version of some other custom field class, inherit from both `EncryptedField` and the other field class:

```
from pgcrypto_expressions.fields import EncryptedField
from somewhere import MyField

class MyEncryptedField(EncryptedField, MyField):
    pass
```

CHAPTER 5

Encryption Key

By default your SECRET_KEY setting is used as the encryption and decryption key. You can override this by setting a PGCRYPTO_KEY setting.

Alternatively, if you are using multiple databases, you can specify a PGCRYPTO_KEY per database in your DATABASES setting. For example:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'PGCRYPTO_KEY': 'super_secret_key',
        ...
    },
    'secondary': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'PGCRYPTO_KEY': 'totally_different_secret_key',
        ...
    },
}
```

Warning: Since encryption is performed on the database server, your encryption key is sent to the database server with each query involving an encrypted field. In order to protect your key, you should only connect to your database with a TLS-protected connection. It is possible that your key could be exposed to an attacker with access to the database server via the pg_stat_activity table or query logs.
For an encrypted-fields solution that encrypts and decrypts on the application side to avoid this problem, see django-fernet-fields.

CHAPTER 6

Indexing, constraints, lookups, and ordering

One advantage of encrypting and decrypting within the database is that encrypted fields may still be used in any type of lookup or database expression, and queries may be ordered by an encrypted field.

However, indexing an encrypted field is not possible without storing the decryption key in the index expression (defeating the value of the encryption), so while lookups can be made against encrypted fields, those lookups or orderings cannot be indexed, meaning their performance will degrade with the size of the table.

Similarly, unique or check constraints can't be applied to encrypted fields.

CHAPTER 7

Contributing

See the contributing docs.