
Django Crypto Extensions

Release dev

Marczak

May 26, 2021

CONTENTS:

- 1 Requirements** **1**
- 2 Installation** **3**
- 3 Usage** **5**
 - 3.1 Django Models CryptoField Package 5
 - 3.2 CryptoBinaryField and CryptoCharField 6
- 4 CONTRIBUTION** **7**
- 5 Indices and tables** **9**

REQUIREMENTS

Django Crypto Extensions requires a supported Django version and runs on Python versions 3.6 and above.

Refer to the project source code repository in [GitHub](#) and see the [Tox configuration](#) and [Python package definition](#) to check if your Django and Python version are supported.

INSTALLATION

Django Crypto Extensions is easy to install from the PyPI package:

```
$ pip install django-crypto-extensions
```

After installing the package, the project settings need to be configured.

1. Add `django_crypto_extensions` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    # Django Crypto Extensions app can be in any position in the INSTALLED_APPS list.  
    'django_crypto_extensions',  
]
```


3.1 Django Models CryptoField Package

A set of generic common Django Fields that automatically encrypt data for database and encrypt for the purpose of usage in Django:

- `CryptoFieldMixin` - Mixin that implement encrypt/decrypt methods.

Example of how to use to create custom `CryptoTextField`:

```
# models.py

from django.db import models

from django_crypto_extensions.django_fields import *

class CustomCryptoTextField(CryptoFieldMixin, models.TextField):
    pass
```

- `CryptoTextField` - `TextField` inheriting Field
- `CryptoCharField` - `CharField` inheriting Field
- `CryptoEmailField` - `EmailField` inheriting Field
- `CryptoIntegerField` - `IntegerField` inheriting Field
- `CryptoDateField` - `DateField` inheriting Field
- `CryptoDateTimeField` - `DateTimeField` inheriting Field
- `CryptoBigIntegerField` - `BigIntegerField` inheriting Field
- `CryptoPositiveIntegerField` - `PositiveIntegerField` inheriting Field
- `CryptoPositiveSmallIntegerField` - `PositiveSmallIntegerField` inheriting Field
- `CryptoSmallIntegerField` - `SmallIntegerField` inheriting Field

Settings

- each `CryptoField` has 2 kwargs `salt_settings_env` and `password`.
- `salt_settings_env` - name of variable stored in `settings.py` file, which will be used as cryptographic salt. default: `salt_settings_env = 'SECRET_KEY'` if settings not set or no `SECRET_KEY` in setting default: `salt = "Salt123!!!"`
- `password` - password to be used in encryption process of given field (together with salt set globally) default = `'password'`

Example:

```
# models.py

from django.db import models

from django_crypto_extensions.django_fields import CryptoEmailField

class TestCryptoEmail(models.Model):

    value = CryptoEmailField(salt_settings_env='NEW_SECRET_KEY', password='new_password')
```

3.2 CryptoBinaryField and CryptoCharField

- A django-rest-framework fields for handling encryption through serialisation. Inputs are String object and internal python representation is Binary object for CryptoBinaryField and String object for CryptoCharField
- It takes the optional parameter salt (Django SECRET_KEY imported from setting as default). If set it use custom cryptographic salt
- It takes the optional parameter password (“Non_nobis!solum?nati!sumus” as default). If set it use a custom password in encryption. **It is highly recommended to use custom one!!**
- It takes the optional parameter ttl (None as default). If set it manage the number of seconds old a message may be for it to be valid. If the message is older than ttl seconds (from the time it was originally created) field will return None and encrypted message will not be enabled for decryption.

Example:

```
from rest_framework import serializers
from drf_extra_fields.crypto_fields import CryptoCharField

class CryptoSerializer(serializers.Serializer):
    crypto_char = CryptoCharField()
```

Example with parameters + It takes custom salt and password. Once saved it will be available for decryption for 1000 seconds.:

```
from rest_framework import serializers
from drf_extra_fields.crypto_fields import CryptoCharField

class CryptoSerializer(serializers.Serializer):
    crypto_char = CryptoCharField(salt="custom salt", password="custom password", ttl=1000)
```

CONTRIBUTION

TESTS

Make sure that you add the test for contributed field to `test/*.py` and run with command before sending a pull request:

```
$ pip install tox # if not already installed
$ tox
```

Or, if you prefer using Docker (recommended):

```
docker build -t django_crypto_extensions .
docker run -v $(pwd):/app -it django_crypto_extensions /bin/bash
tox
```


INDICES AND TABLES

- genindex
- modindex
- search