fcrypter Documentation

Release 0.1.0

John Paraskevopoulos

May 02, 2017

Contents

Contents:

CHAPTER 1

fcrypter

File encryption made simple

- Free software: GNU General Public License v3
- Documentation: https://fcrypter.readthedocs.io.

Features

- TODO

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Stable release

To install fcrypter, run this command in your terminal:

```
$ pip install fcrypter
```

This is the preferred method to install fcrypter, as it will always install the most recent stable release.

If you don’t have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for fcrypter can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ioparaskev/fcrypter
```

Or download the tarball:

```
$ curl  -OL https://github.com/ioparaskev/fcrypter/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use fcrypter in a project:

```
import fcrypter
```

CHAPTER 4

fcrypter package

Subpackages

fcrypter.base package

Submodules

fcrypter.base.custom_exceptions module

exception `fcrypter.base.custom_exceptions.`**`WrongKeySizeError`**

Bases: `exceptions.Exception`

fcrypter.base.templates module

class `fcrypter.base.templates.`**`Cryptor`**(*supported_sizes*)

Bases: `object`

`decrypt_file`(*key*, *encr_fpath*, *decr_fpath*)

File decryption base method

Needs a key, a file path to decrypt and a resulting file path for the decrypted file to be saved

Parameters

- **`key`** – a key to use for the file decryption
- **`encr_fpath`** – absolute path for the encrypted file
- **`decr_fpath`** – absolute path for the decrypted file to be saved

`encrypt_file`(*key*, *f_path*, *encr_fpath*)

File encryption base method

Needs a key, a file path to encrypt and a resulting file path for the encrypted file to be saved

Parameters

- **key** – a key to use for the file encryption
- **f_path** – absolute path for the file to be encrypted
- **encr_fpath** – absolute path for the encrypted file to be saved

generate_random_password(*size=None*, *chars=None*)

Random password generator If size not specified, chooses randomly from the supported sizes If chars are specified, generates password for those chars Else generates random size bytes

Parameters

- **size** (*int*) – password size
- **chars** (*iterable*) – chars to choose password from

Returns random password

Return type bytes

verify_supported_key_size(*key*)

Verifies key size is supported

Parameters **key** – key to be used for encryption

Module contents

Submodules

fcrypter.aes module

class fcrypter.aes.**AESCryptorEAX**

Bases: *fcrypter.base.templates.Cryptor*

AES EAX mode file encryption through use of pycryptodome AES library

For details, see https://en.wikipedia.org/wiki/EAX_mode http://web.cs.ucdavis.edu/~rogaway/papers/eax.pdf

decrypt_file(*key*, *encr_fpath*, *decr_fpath*)

Decrypts file and verifies integrity

encrypt_file(*key*, *f_path*, *encr_fpath*)

Encrypts files, then digests. Provides plaintext & ciphertext integrity

More info about "Encrypt then MAC" in Bellare & Namprempre paper:

> Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm
>
> http://cseweb.ucsd.edu/~mihir/papers/oem.pdf

Since AES library doesn't provide easy way to do this in seperate actions we add 16 dummy bytes as MAC, write the encrypted content and then seek the dummy bytes and overwrite with the correct MAC

This allows us to read a file for encryption in small chunks to avoid insufficient memory errors

fcrypter.cli module

fcrypter.fcrypter module

`fcrypter.fcrypter.`**`get_aes_supported_modes`**`()`

Returns a named tuple with supported AES cipher modes and their class name callback

`fcrypter.fcrypter.`**`get_supported_ciphers`**`()`

Returns a list of named tuples with supported cipher name and callback class name :rtype: dict

`fcrypter.fcrypter.`**`pad_key`**`(`*key*, *cipher*`)`

Adds extra characters to a key to reach the maximum supported key length :param key: encryption key :type cipher: Cryptor :return: str

Module contents

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/ioparaskev/fcrypter/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

fcrypter could always use more documentation, whether as part of the official fcrypter docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ioparaskev/fcrypter/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *fcrypter* for local development.

1. Fork the *fcrypter* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/fcrypter.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv fcrypter
   $ cd fcrypter/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 fcrypter tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/ioparaskev/fcrypter/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_fcrypter
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index