pynoteslib

Release 0.5.0

Ian Stanley

Aug 24, 2021

CONTENTS

CHAPTER

ONE

PYNOTESLIB

PyNoteslib is a library of functions and classes to assist in building apps to manage GPG encrypted notes.

It is based upon an earlier project of mine *Standard unix Notes* which was a set of bourne shell scripts that implemented an easy way to manage gpg encrypted notes.

Pynoteslib follows the same structure

1.1 Overview

docs	
tests	
package	

Encrypted Library

- Free software: MIT license

1.2 Installation

```
pip install pynoteslib
```

You can also install the in-development version with:

```
pip install https://github.com/Standard-Unix-Notes/pynoteslib/archive/master.zip
```

1.3 Documentation

The documentation for the library is hosted on ReadTheDocs.io at https://pynoteslib.readthedocs.io/

1.4 Development

Contributions and pull requests are welcome: see the documentation for details.

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`Currently not available for Windows`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INSTALLATION

At the command line:

```
pip install pynoteslib
```

CHAPTER

THREE

USAGE

To use pynoteslib in a project:

```
import pynoteslib
```

CHAPTER

FOUR

REFERENCE

4.1 pynoteslib package

4.1.1 pynoteslib module

PYNOTESLIB the Python library implementation of Standard Unix Notes.

It implements the notes() class and a number of functions to manipulate notebooks and configuration.

NOTES allows the user to have multiple notebooks and even a default notebook. The initial notebook is called simply 'Notes' and all notes created or imported will default to this notebook.

The user may create additional notebooks at any time and choose to USE a preferred notebook where all future notes will be created until the user chooses to USE another notebook. The user can quickly switch back to a DEFAULT notebook by not specifying which notebook to USE.

Full documentation can be found at https://pynoteslib.readthedocs.io/en/latest/

class pynoteslib.**Notes**

Bases: object

Object for managing a noteand it's plaintext/ciphertext

Variables

- **title** – title of note
- **filename** – filename of note
- **fullpath** – full pathname of file containing note
- **ciphertext** – string containing the ciphertext of note
- **plaintext** – string containing the plaintext of note

NB only one of either ciphertext or plaintext should be set at any time.

Notes class constructor. Do not use directly use one of the following functions:

load_note_from_file(note_filename) note_from_ciphertext(str) note_from_plaintext(str) import_note_from_file(filename)

add_extension()

Appends '.asc' to the basename of self.filename

Param none

Returns none

property ciphertext
ciphertext property of note

decrypt()
Encrypts self.plaintext -> selfciphertext and resets self.plaintext

Returns self.ciphertext

Return type str

encrypt()
Encrypts self.plaintext -> selfciphertext and resets self.plaintext

Returns self.ciphertext

Return type str

property filename
filename property of note

get_extension()
Returns extension of self.filename

Param none

Returns self.filename's extension

Return type str

import_from_file()
Loads plaintext from file self.filename (fullpath)

Param none

Returns none

is_encrypted()
Check if note is encrypted

Param none

Returns True if self.ciphertext != ''

load_ciphertext()
Loads ciphertext from file self.filename

Param none

Returns none

load_plaintext()
Loads plaintext from file self.filename

Param none

Returns none

property plaintext
plaintext property of note

remove_extension()
Removes extension from self.filename

Param none

Returns none

save_ciphertext()
Saves Ciphertext of note to file named self.filename adding the extension '.asc'

Param none

Returns none

save_plaintext()
Saves Plaintext of note to file named self.filename

Param none

Returns none

property title
title property of note

pynoteslib.**backup**(*conf*)
Backup configuration, notes and notebook to tar file in the directory above the NOTESDIR (default = HOME)

Param none

Returns The return code of tarfile creation/write

Return type bool

pynoteslib.**change_spaces**(*string*)
Returns a string with all spaces in 'string' have been replaced with '_'

Parameters **string** – String to have spaces replaced

Type str

Returns Supplied 'string' with spaces replaced with '_'

Return type str

pynoteslib.**config_file_exists**()
Checks to see if NOTESDIR/config file exists

Param none

Returns True if NOTESDIR/config file exists

Return type bool

pynoteslib.**copy_to_notebook**(*filename*, *notebook*)
Copies note from current USE'd notebook to another notebook

Parameters

- **filename** (*str*) – The filename of note to be copied
- **notebook** (*str*) – The target notebook name

Returns True on successful copy

Return type bool

pynoteslib.**create_config**()
Create directory structure under NOTESDIR and TOML config file NOTESDIR/config

Param none

Returns none

pynoteslib.**create_notebook**(*title*)
Create a notebook with foldername 'title'

Parameters **title** (*str*) – title of notebook

Returns True on successful creation of notebook's folder

Return type bool

pynoteslib.**default_notebook**(*notebook*)
Set a notebook as th edefault notebook (use_notebook() defaults to the DEFAULT notebook if '' instead of a notebook title)

Parameters **notebook** (*str*) – notebook to set as default

Returns Returns True on success of write_config() with updated configuration

Return type bool

pynoteslib.**delete_note**(*filename*)
Deletes a note on disk inside the currently USE'd notebook

Parameters **filename** (*str*) – A string containing the filename of note to be deleted

Returns True on successful deletion of note

Return type bool

pynoteslib.**delete_notebook**(*title*)
Deletes an existing notebook oldtitle and included notes

Parameters **title** (*str*) – Title of existing notebook

Returns True on successful deletion of notebook's folder

Return type bool

pynoteslib.**duplicate_note**(*oldname*, *newname*)
Duplicates an encrypted note on disk inside the currently USE'd notebook

Parameters

- **oldname** (*str*) – The new filename for note
- **newname** (*str*) – The new filename for note

Returns True on successful rename of note

Return type bool

pynoteslib.**duplicate_notebook**(*oldtitle*, *newtitle*)
Duplicates an existing notebook oldtitle as newtitle with all notes duplicated.

Parameters

- **oldtitle** (*str*) – Title of existing notebook
- **newtitle** (*str*) – New Title for notebook

Returns True on successful duplication of notebook's folder

Return type bool

pynoteslib.**get_config**()
Reads configuration from the TOML file NOTESDIR/config. If 'config' file does not exist, calls create_config() to create

Param none

Returns Configuration loaded from the TOML file 'config'

Return type dict

pynoteslib.**get_config_file**()
Get the fullpath to the app configuration file NOTESDIR/config

Param none

Returns fullpath to the config file fullpath

Return type str

pynoteslib.**get_default_gpg_key**()
Locates the first private key in the users GPG keyring

Under testing conditions it returns the test@pynoteslib GPG key shown in _default_config['gpgkey'] to use in testing

In normal conditions it returns the first private gpgkey found in the user's keyring

Param none

Returns The first GPG key ID found in user's keyring

Return type str

pynoteslib.**get_default_notebook**()
Reads config file and returns what notebook is the default

Param none

Returns The name of the default notebook

Return type str

pynoteslib.**get_fullpath**(*name*)
Return full pathname of passed parameter

Parameters **name** (*str*) – A notebook, filename (eg. 'config') or expression`

Returns Returns full path for 'name' UNDER the NOTESDIR

Return type str

pynoteslib.**get_note_fullpath**(*note*, *notebook=''*)
Returns the full pathname of a note within the currently USE'd Notebook

Parameters **note** – The title (or filename) of a note

Type str

Returns Returns full path to a note

Return type str

pynoteslib.**get_notebooks**()
Returns a list of all notebooks in NOTESDIR

Param none

Returns A list[] of notebooks

Return type list

pynoteslib.**get_notes**(*notebook=''*)
Returns a list of note in given notebook (or the USE'd notebook)

Parameters **notebook** (*str, optional*) – Specified notebook to USE, defaults to DEFAULT notebook

Returns list of notes in notebook; or [] for invalid notebook

Return type list

pynoteslib.**get_notesdir**()
Gets the fullpath to the main app directory

Param none

Returns the app's home folder (either NOTESDIR or $HOME/.notes)

Return type str

pynoteslib.**get_use_notebook**()
Reads config file and returns what notebook is currently used notebook

Param none

Returns The currently 'use'd notebook (where notes will be created)

Return type str

pynoteslib.**import_note_from_file**(*filename*)
Imports note from file

Parameters **filename** (*str*) – filename to be imported

Returns note

Return type class

pynoteslib.**load_note_from_file**(*filename*)
Opens file and assigns contents to plaintext or ciphertext

Parameters **filename** (*str*) – fullpath of filename

Returns returns success or failure

Return type bool

pynoteslib.**move_to_notebook**(*filename*, *notebook*)
Moves a note from the currently USE'd notebook to another notebook

Parameters

- **filename** (*str*) – The filename to move
- **notebook** – The target notebook name

Returns True on successful move of note to notebook

Return type bool

pynoteslib.**new_key**(*newkey*)
Change encryption key for all notes. Traverses filesystem in NOTESDIR/[all notebooks]. Decrypts and re-encrypts with specified newkey

Parameters **newkey** (*str*) – New valid gpg privakey keyid

Returns Returns True on re-encryption; False on invalid private key

Return type bool

pynoteslib.**note_from_ciphertext**(*ciphertext*)
Creates note from supplied ciphertext

Parameters **ciphertext** (*str*) – ciphertext of note

Returns note

Return type class

pynoteslib.**note_from_plaintext**(*plaintext*)
Creates note from supplied plaintext

Parameters **plaintext** (*str*) – plaintext of note

Returns note

Return type class

pynoteslib.**pynoteslib_version**()
Returns version no of library

pynoteslib.**rename_note**(*oldname*, *newname*)
Renames a note on disk inside the currently USE'd notebook

Parameters

- **oldname** (*str*) – The old filename for note
- **newname** (*str*) – The new filename for note

Returns True on sucessful renaming of note

Return type bool

pynoteslib.**rename_notebook**(*oldtitle*, *newtitle*)
Renames existing notebook oldtitle as newtitle

Parameters

- **oldtitle** (*str*) – Title of existing notebook
- **newtitle** (*str*) – New Title for notebook

Returns True on successful rename of notebook's folder

Return type bool

pynoteslib.**use_notebook**(*notebook=''*)
Reads config file and returns the DEFAULT notebook. If no notebook is specified then the USE notebook is set to the DEFAULT notebook

Parameters **notebook** (*str*) – Title of notebook to USE, optional

Returns Returns True on successful write of new config file

Return type bool

pynoteslib.**validate_gpg_key**(*gpgkeyid*)
Validates the specified gpgkeyid is a private key in the user's keyring

Param none

Returns True if gpgkey is a valid private key

Return type bool

pynoteslib.**write_config**(*conf*)
Writes app configuration to TOML file NOTESDIR/config (see _default_config as a sample structure)

Parameters **conf** – Dictionary containing configuration data

Type dict

Returns True on successful write of configfile

Return type bool

CHAPTER

FIVE

CONTRIBUTING

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

5.1 Bug reports

When 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Standard-Unix-Notes/pynoteslib/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 code contributions are welcome :)

5.4 Development

To set up *pynoteslib* for local development:

1. Fork pynoteslib (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/pynoteslib.git
```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER

SIX

PYNOTES & THE TEST GPG KEYS

6.1 GPG keys used in the pytest testing suite

The test suite GPG keys can be found in the gpgkeys directory and should be imported into the developers keyring prior to running the PYTEST test suite.

Without importing and marking them as trusted GPG will fail to use them for decrypting during testing (GPG will prompt for use anyway but this will break the tests and fail the assertions used afterwards).

6.2 Importing the test gpg keys

To import the test gpgkeys:

```
$ gpg --import gpgkeys/\*.asc
```

6.3 Changing the gpg trust level for the test keys

You will then need to change the trust level:

```
$ gpg -K
```

and then for each of the test@pynotes.lib and alttest@pynotes.lib run the following to mark the test gpg keys as trusted:

```
$ gpg --edit-key <uid>

gpg> trust

Please decide how far you trust this user to correctly verify other
users' keys (by looking at passports, checking fingerprints from
different sources, etc.)

    1 = I don't know or won't say
    2 = I do NOT trust
    3 = I trust marginally
    4 = I trust fully
    5 = I trust ultimately
```

(continues on next page)

(continued from previous page)

```
    m = back to the main menu

Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y

Please note that the shown key validity is not necessarily correct
unless you restart the program.

gpg> quit
```

These two keys are only used in the pytest test suite for PYNOTESLIB and are not used elsewhere so it is safe to mark these as trust ultimately.

6.4 Pytest encryption errors

Without marking the gpg keys as trusted the GPG decryption will fail and the new_key test will crash:

```
_____________________________ test_new_key _____________________________

    def test_new_key():
        conf = nl.get_config()
        print(conf['gpgkey'])

        # Create a note with TESTKEY1 (default in unittest)
        message = "This is some text to test new_key()"
        n1 = nl.Notes(title='testing newkey')
        n1.set_plaintext(message)
        ct = n1.encrypt()
        n1.save_ciphertext()
        assert  os.path.exists(nl.get_note_fullpath(n1.filename))

        # change all the notes to TESTKEY2
        assert nl.new_key(TESTKEY2)

        # import same key into new Notes object and decrypt
        n2 = nl.Notes(filename='testing_newkey.asc')
        print(f"n2 => {n2}")
>       assert n2.decrypt() == message
E       AssertionError: assert '' == 'This is some...est new_key()'
E         - This is some text to test new_key()

tests/notes_class/test_new_key.py:27: AssertionError


----------------------------- Captured log call ------------------------------
WARNING  gnupg:gnupg.py:1015 gpg returned a non-zero error code: 2
WARNING  gnupg:gnupg.py:1015 gpg returned a non-zero error code: 2
WARNING  gnupg:gnupg.py:1015 gpg returned a non-zero error code: 2
```

(continues on next page)

(continued from previous page)

```
========================= short test summary info ==========================
FAILED tests/notes_class/test_new_key.py::test_new_key - AssertionError:
assert '' == 'This is some...est n...
======================== 1 failed, 27 passed in 2.80s ========================
```

CHAPTER

SEVEN

AUTHORS

- Ian Stanley - https://github.com/iandstanley

CHAPTER

EIGHT

CHANGELOG

8.1 0.1.0 (2021-08-08)

- First release on PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

V

W