Python OMEMO Library

Release 0.1.0

Apr 24, 2017

Contents

CHAPTER 1

Overview

docs	
tests	
package	

This is an implementation **OMEMO Multi-End Message and Object Encryption** in Python.

Installation

```
pip install python-omemo
```

Documentation

https://python-omemo.readthedocs.org/

Development

To set up *python-omemo* for local development:

1. Fork python-omemo on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-omemo.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. Run all the checks, doc builder and spell checker with tox one command:

   ```
   tox
   ```

Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

Contributing

The **Python OMEMO** project direction is the sum of documented problems: everybody is invited to describe and discuss a problem in the issue tracker. Contributed solutions

encourage participation.

Some problem fields we initially focus on are:

- Creation of a reusable python omemo implementation
- Reusability bu the Gajim OMEMO plugin

CHAPTER 2

Installation

At the command line:

```
pip install python-omemo
```

CHAPTER 3

Usage

To use Python OMEMO Library in a project:

```
import omemo
```

CHAPTER 4

Reference

OmemoState

class `omemo.state.`**`OmemoState`**(*connection*)

`__init__`(*connection*)
Instantiates an OmemoState object.

Parameters **`connection`** – an `sqlite3.Connection`

`__module__` **= 'omemo.state'**

`add_devices`(*name*, *devices*)
Return a an.

Parameters

- **jid** (*string*) – The contacts jid
- **devices** (*[int]*) – A list of devices

`add_own_devices`(*devices*)
Overwrite the current **:py:attribute:‘OmemoState.own_devices‘** with the given devices.

Parameters **devices** (*[int]*) – A list of device_ids

`build_session`(*recipient_id*, *device_id*, *bundle_dict*)

`bundle`

`create_msg`(*from_jid*, *jid*, *plaintext*)

`decrypt_msg`(*msg_dict*)

`device_ids` **= {}**

`device_list_for`(*jid*)
Return a list of known device ids for the specified jid.

Parameters **jid** (*string*) – The contacts jid

devices_without_sessions (*jid*)
List device_ids for the given jid which have no axolotl session.

Parameters **jid** (*string*) – The contacts jid

Returns *[int]* – A list of device_ids

encryption = None

get_session_cipher (*jid*, *device_id*)

handlePreKeyWhisperMessage (*recipient_id*, *device_id*, *key*)

handleWhisperMessage (*recipient_id*, *device_id*, *key*)

own_device_id

own_device_id_published ()
Return *True* only if own device id was added via **:py:method:‘OmemoState.add_own_devices()‘**.

own_devices = []

own_devices_without_sessions (*own_jid*)
List own device_ids which have no axolotl session.

Parameters **own_jid** (*string*) – Workaround for missing own jid in OmemoState

Returns *[int]* – A list of device_ids

session_ciphers = {}

CHAPTER 5

Collective Code Construction Contract

The **Collective Code Construction Contract (C4)** is an evolution of the github.com Fork + Pull Model, aimed at providing an optimal collaboration model for free software projects. This is revision 1 of the C4 specification.

License

Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119.

Goals

C4 is meant to provide a reusable optimal collaboration model for open source software projects. It has these specific goals:

- To maximize the scale of the community around a project, by reducing the friction for new Contributors and creating a scaled participation model with strong positive feedbacks;
- To relieve dependencies on key individuals by separating different skill sets so that there is a larger pool of competence in any required domain;
- To allow the project to develop faster and more accurately, by increasing the diversity of the decision making process;
- To support the natural life cycle of project versions from experimental through to stable, by allowing safe experimentation, rapid failure, and isolation of stable code;
- To reduce the internal complexity of project repositories, thus making it easier for Contributors to participate and reducing the scope for error;
- To enforce collective ownership of the project, which increases economic incentive to Contributors and reduces the risk of hijack by hostile entities.

Design

Preliminaries

- The project **SHALL** use the git distributed revision control system.
- The project **SHALL** be hosted on github.com or equivalent, herein called the “Platform”.
- The project **SHALL** use the Platform issue tracker.
- The project **SHOULD** have clearly documented guidelines for code style.
- A “Contributor” is a person who wishes to provide a patch, being a set of commits that solve some clearly identified problem.
- A “Maintainer” is a person who merges patches to the project. Maintainers are not developers; their job is to enforce process.
- Contributors **SHALL NOT** have commit access to the repository unless they are also Maintainers.
- Maintainers **SHALL** have commit access to the repository.
- Everyone, without distinction or discrimination, **SHALL** have an equal right to become a Contributor under the terms of this contract.

Licensing and Ownership

- The project **SHALL** use a share-alike license, such as the GPLv3 or a variant thereof (LGPL, AGPL), or the MPLv2.
- All contributions to the project source code (“patches”) **SHALL** use the same license as the project.
- All patches are owned by their authors. There **SHALL NOT** be any copyright assignment process.
- The copyrights in the project **SHALL** be owned collectively by all its Contributors.
- Each Contributor **SHALL** be responsible for identifying themselves in the project Contributor list.

Patch Requirements

- Maintainers and Contributors **MUST** have a Platform account and **SHOULD** use their real names or a well-known alias.
- A patch **SHOULD** be a minimal and accurate answer to exactly one identified and agreed problem.
- A patch **MUST** adhere to the code style guidelines of the project if these are defined.
- A patch **MUST** adhere to the "Evolution of Public Contracts" guidelines defined below.
- A patch **SHALL NOT** include non-trivial code from other projects unless the Contributor is the original author of that code.
- A patch **MUST** compile cleanly and pass project self-tests on at least the principle target platform.
- A patch commit message **SHOULD** consist of a single short (less than 50 character) line summarizing the change, optionally followed by a blank line and then a more thorough description.
- A "Correct Patch" is one that satisfies the above requirements.

Development Process

- Change on the project **SHALL** be governed by the pattern of accurately identifying problems and applying minimal, accurate solutions to these problems.
- To request changes, a user **SHOULD** log an issue on the project Platform issue tracker.
- The user or Contributor **SHOULD** write the issue by describing the problem they face or observe.
- The user or Contributor **SHOULD** seek consensus on the accuracy of their observation, and the value of solving the problem.
- Users **SHALL NOT** log feature requests, ideas, suggestions, or any solutions to problems that are not explicitly documented and provable.
- Thus, the release history of the project **SHALL** be a list of meaningful issues logged and solved.
- To work on an issue, a Contributor **SHALL** fork the project repository and then work on their forked repository.
- To submit a patch, a Contributor **SHALL** create a Platform pull request back to the project.
- A Contributor **SHALL NOT** commit changes directly to the project.
- If the Platform implements pull requests as issues, a Contributor **MAY** directly send a pull request without logging a separate issue.
- To discuss a patch, people **MAY** comment on the Platform pull request, on the commit, or elsewhere.
- To accept or reject a patch, a Maintainer **SHALL** use the Platform interface.
- Maintainers **SHOULD NOT** merge their own patches except in exceptional cases, such as non-responsiveness from other Maintainers for an extended period (more than 1-2 days).
- Maintainers **SHALL NOT** make value judgments on correct patches.
- Maintainers **SHALL** merge correct patches from other Contributors rapidly.
- The Contributor **MAY** tag an issue as "Ready" after making a pull request for the issue.
- The user who created an issue **SHOULD** close the issue after checking the patch is successful.
- Maintainers **SHOULD** ask for improvements to incorrect patches and **SHOULD** reject incorrect patches if the Contributor does not respond constructively.

- Any Contributor who has value judgments on a correct patch **SHOULD** express these via their own patches.
- Maintainers **MAY** commit changes to non-source documentation directly to the project.

Creating Stable Releases

- The project **SHALL** have one branch ("master") that always holds the latest in-progress version and **SHOULD** always build.
- The project **SHALL NOT** use topic branches for any reason. Personal forks **MAY** use topic branches.
- To make a stable release someone **SHALL** fork the repository by copying it and thus become maintainer of this repository.
- Forking a project for stabilization **MAY** be done unilaterally and without agreement of project maintainers.
- A stabilization project **SHOULD** be maintained by the same process as the main project.
- A patch to a stabilization project declared "stable" **SHALL** be accompanied by a reproducible test case.

Evolution of Public Contracts

- All Public Contracts (APIs or protocols) **SHALL** be documented.
- All Public Contracts **SHOULD** have space for extensibility and experimentation.
- A patch that modifies a stable Public Contract **SHOULD** not break existing applications unless there is overriding consensus on the value of doing this.
- A patch that introduces new features to a Public Contract **SHOULD** do so using new names.
- Old names **SHOULD** be deprecated in a systematic fashion by marking new names as "experimental" until they are stable, then marking the old names as "deprecated".
- When sufficient time has passed, old deprecated names **SHOULD** be marked "legacy" and eventually removed.
- Old names **SHALL NOT** be reused by new features.
- When old names are removed, their implementations **MUST** provoke an exception (assertion) if used by applications.

Project Administration

- The project founders **SHALL** act as Administrators to manage the set of project Maintainers.
- The Administrators **SHALL** ensure their own succession over time by promoting the most effective Maintainers.
- A new Contributor who makes a correct patch **SHALL** be invited to become a Maintainer.
- Administrators **MAY** remove Maintainers who are inactive for an extended period of time, or who repeatedly fail to apply this process accurately.
- Administrators **SHOULD** block or ban "bad actors" who cause stress and pain to others in the project. This should be done after public discussion, with a chance for all parties to speak. A bad actor is someone who repeatedly ignores the rules and culture of the project, who is needlessly argumentative or hostile, or who is offensive, and who is unable to self-correct their behavior when asked to do so by others.

Further Reading

- Argyris' Models 1 and 2 - the goals of C4.1 are consistent with Argyris' Model 2.
- Toyota Kata - covering the Improvement Kata (fixing problems one at a time) and the Coaching Kata (helping others to learn the Improvement Kata).

Implementations

- The ZeroMQ community uses the C4.1 process for many projects.
- OSSEC uses the C4.1 process.
- The Machinekit community uses the C4.1 process.

CHAPTER 6

Authors

- Bahtiar *kalkin-* Gadimov - https://github.com/kalkin
- Daniel Gultsch - https://github.com/inputmice
- Tarek Galal - https://github.com/tgalal (original axolotl store implementation)

CHAPTER 7

Changelog

0.1.0 (2016-01-11)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index