Python OMEMO Library

Release 0.1.0

Aug 29, 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

XEP: OMEMO Encryption

Abstract:: This specification defines a protocol for end-to-end encryption in one-on-one chats that may have multiple clients per account.

Status:: ProtoXEP

Type:: Standards Track

Version:: 0.0.1

Last Updated: 2015-10-25

Warning: WARNING: This document has not yet been accepted for consideration or approved in any official manner by the XMPP Standards Foundation, and this document is not yet an XMPP Extension Protocol (XEP). If this document is accepted as a XEP by the XMPP Council, it will be published at <https://xmpp.org/extensions/> and announced on the <standards@xmpp.org> mailing list.

1. Introduction

1.1 Motivation

There are two main end-to-end encryption schemes in common use in the XMPP ecosystem, Off-the-Record (OTR) messaging (Current Off-the_Record Messaging Usage (XEP-0364)) and OpenPGP (Current Jabber OpenPGP Usage (XEP-0027)). OTR has significant usability drawbacks for inter-client mobility. As OTR sessions exist between exactly two clients, the chat history will not be synchronized across other clients of the involved parties. Furthermore, OTR chats are only possible if both participants are currently online, due to how the rolling key agreement scheme of OTR works. OpenPGP, while not suffering from these mobility issues, does not provide any kind of forward secrecy and is vulnerable to replay attacks. Additionally, PGP over XMPP uses a custom wireformat which is defined by convention rather than standardization, and involves quite a bit of external complexity.

This XEP defines a protocol that leverages axolotl encryption to provide multi-end to multi-end encryption, allowing messages to be synchronized securely across multiple clients, even if some of them are offline.

1.2 Overview

The general idea behind this protocol is to maintain separate, long-standing axolotl-encrypted sessions with each device of each contact (as well as with each of our other devices), which are used as secure key transport channels. In this scheme, each message is encrypted with a fresh, randomly generated encryption key. An encrypted header is added to the message for each device that is supposed to receive it. These headers simply contain the key that the payload message is encrypted with, and they are seperately encrypted using the session corresponding to the counterpart device. The encrypted payload is sent together with the headers as a <message> stanza. Individual recipient devices can decrypt the header item intended for them, and use the contained payload key to decrypt the payload message.

As the encrypted payload is common to all recipients, it only has to be included once, reducing overhead. Furthermore, axolotl's transparent handling of messages that were lost or received out of order, as well as those sent while the recipient was offline, is maintained by this protocol. As a result, in combination with Message Carbons (XEP-0280) and Message Archive Management (XEP-0313), the desired property of inter-client history synchronization is achieved.

OMEMO version 0 uses v3 messages of the axolotl protocol. Instead of an axolotl key server, PEP (Personal Eventing Protocol (XEP-0163)) is used to publish key data.

2. Requirements

- Provide forward secrecy
- Ensure chat messages can be deciphered by all (capable) clients of both
- parties
- Be usable regardless of the participants' online statuses
- Provide a method to exchange auxilliary keying material. This
- could for example be used to secure encrypted file transfers.

3. Glossary

3.1 General Terms

Device:: A communication end point, i.e. a specific client instance

OMEMO element:: An <*encrypted*> element in the *urn:xmpp:omemo:0* namespace. Can be either MessageElement or a KeyTransportElement

MessageElement:: An OMEMO element that contains a chat message. Its <*payload*>, when decrypted, corresponds to a <*message*>'s <*body*>.

KeyTransportElement:: An OMEMO element that does not have a <*payload*>. It contains a fresh encryption key, which can be used for purposes external to this XEP.

Bundle:: A collection of publicly accessible data that can be used to build a session with a device, namely its public IdentityKey, a signed PreKey with corresponding signature, and a list of (single use) PreKeys.

rid:: The device id of the intended recipient of the containing <*key*>

sid:: The device id of the sender of the containing OMEMO element

3.2 Axolotl-specific

IdentityKey:: Per-device public/private key pair used to authenticate communications

PreKey:: A Diffie-Hellman public key, published in bulk and ahead of time

PreKeyWhisperMessage:: An encrypted message that includes the initial key exchange. This is used to transparently build sessions with the first exchanged message.

WhisperMessage:: An encrypted message

4. Use Cases

4.1 Setup

The first thing that needs to happen if a client wants to start using OMEMO is they need to generate an IdentityKey and a Device ID. The IdentityKey is a Curve25519 public/private Key pair. The Device ID is a randomly generated integer between 1 and 2^31 - 1.

4.2 Discovering peer support

In order to determine whether a given contact has devices that support OMEMO, the devicelist node in PEP is consulted. Devices MUST subscribe to *'urn:xmpp:omemo:0:devicelist* via PEP, so that they are informed whenever their contacts add a new device. They MUST cache the most up-to-date version of the devicelist.

Example 1. Devicelist update received by subscribed clients

```
<message from='juliet@capulet.lit'
         to='romeo@montague.lit'
         type='headline'
         id='update_01'>
<event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:omemo:0:devicelist'>
    <item>
        <list xmlns='urn:xmpp:omemo:0'>
        <device id='12345' />
        <device id='4223' />
        </list>
    </item>
    </items>
</event>
</message>
```

4.3 Announcing support

In order for other devices to be able to initiate a session with a given device, it first has to announce itself by adding its device ID to the devicelist PEP node.

Example 2. Adding the own device ID to the list

```
<iq from='juliet@capulet.lit' type='set' id='announce1'>
<pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:omemo:0:devicelist'>
    <item>
```

```
        <list xmlns='urn:xmpp:omemo:0'>
        <device id='12345' />
        <device id='4223' />
        <device id='31415' />
        </list>
    </item>
    </publish>
</pubsub>
</iq>
```

This step presents the risk of introducing a race condition: Two devices might simultaneously try to announce themselves, unaware of the other's existence. The second device would overwrite the first one. To mitigate this, devices MUST check that their own device ID is contained in the list whenever they receive a PEP update from their own account. If they have been removed, they MUST reannounce themselves.

Furthermore, a device MUST announce it's IdentityKey, a signed PreKey, and a list of PreKeys in a separate, per-device PEP node. The list SHOULD contain 100 PreKeys, but MUST contain no less than 20.

Example 3. Announcing bundle information

```
<iq from='juliet@capulet.lit' type='set' id='announce2'>
<pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:omemo:0:bundles:31415'>
    <item>
        <bundle xmlns='urn:xmpp:omemo:0'>
        <signedPreKeyPublic signedPreKeyId='1'>
            BASE64ENCODED...
        </signedPreKeyPublic>
        <signedPreKeySignature>
            BASE64ENCODED...
        </signedPreKeySignature>
        <identityKey>
            BASE64ENCODED...
        </identityKey>
        <prekeys>
            <preKeyPublic preKeyId='1'>
            BASE64ENCODED...
            </preKeyPublic>
            <preKeyPublic preKeyId='2'>
            BASE64ENCODED...
            </preKeyPublic>
            <preKeyPublic preKeyId='3'>
            BASE64ENCODED...
            </preKeyPublic>
            <!-- ... -->
        </prekeys>
        </bundle>
    </item>
    </publish>
</pubsub>
</iq>
```

4.4 Building a session

In order to build a session with a device, their bundle information is fetched.

Example 4. Fetching a device's bundle information

```
<iq type='get'
    from='romeo@montague.lit'
    to='juliet@capulet.lit'
    id='fetch1'>
<pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:omemo:0:bundles:31415'/>
</pubsub>
</iq>
```

A random preKeyPublic entry is selected, and used to build an axolotl session.

4.5 Sending a message

In order to send a chat message, its <*body*> first has to be encrypted. The client MUST use fresh, randomly generated key/IV pairs with AES-128 in Galois/Counter Mode (GCM). For each intended recipient device, i.e. both own devices as well as devices associated with the contact, this key is encrypted using the corresponding long-standing axolotl session. Each encrypted payload key is tagged with the recipient device's ID. This is all serialized into a MessageElement, which is transmitted in a <*message*> as follows:

Example 5. Sending a message

```
<message to='juliet@capulet.lit' from='romeo@montague.lit' id='send1'>
<encrypted xmlns='urn:xmpp:omemo:0'>
    <header sid='27183'>
    <key rid='31415'>BASE64ENCODED...</key>
    <key rid='12321'>BASE64ENCODED...</key>
    <!-- ... -->
    <iv>BASE64ENCODED...</iv>
    </header>
    <payload>BASE64ENCODED</payload>
</encrypted>
<store xmlns='urn:xmpp:hints'/>
</message>
```

4.6 Sending a key

The client may wish to transmit keying material to the contact. This first has to be generated. The client MUST generate a fresh, randomly generated key/IV pair. For each intended recipient device, i.e. both own devices as well as devices associated with the contact, this key is encrypted using the corresponding long-standing axolotl session. Each encrypted payload key is tagged with the recipient device's ID. This is all serialized into a KeyTransportElement, omitting the <*payload*> as follows:

Example 6. Sending a key

```
<encrypted xmlns='urn:xmpp:omemo:0'>
<header sid='27183'>
    <key rid='31415'>BASE64ENCODED...</key>
    <key rid='12321'>BASE64ENCODED...</key>
    <!-- ... -->
    <iv>BASE64ENCODED...</iv>
</header>
</encrypted>
```

This KeyTransportElement can then be sent over any applicable transport mechanism.

4.7 Receiving a message

When an OMEMO element is received, the client MUST check whether there is a <*key*> element with an rid attribute matching its own device ID. If this is not the case, the element MUST be silently discarded. If such an element exists, the client checks whether the element's contents are a PreKeyWhisperMessage.

If this is the case, a new session is built from this received element. The client SHOULD then republish their bundle information, replacing the used PreKey, such that it won't be used again by a different client. If the client already has a session with the sender's device, it MUST replace this session with the newly built session. The client MUST delete the private key belonging to the PreKey after use.

If the element's contents are a WhisperMessage, and the client has a session with the sender's device, it tries to decrypt the WhisperMessage using this session. If the decryption fails or if the element's contents are not a WhisperMessage either, the OMEMO element MUST be silently discarded.

If the OMEMO element contains a <*payload*>, it is an OMEMO message element. The client tries to decrypt the base 64 encoded contents using the key extracted from the <*key*> element. If the decryption fails, the client MUST silently discard the OMEMO message. If it succeeds, the decrypted contents are treated as the <*body*> of the received message.

If the OMEMO element does not contain a <*payload*>, the client has received a KeyTransportElement. The key extracted from the <*key*> element can then be used for other purposes (e.g. encrypted file transfer).

5. Business Rules

Before publishing a freshly generated Device ID for the first time, a device MUST check whether that Device ID already exists, and if so, generate a new one.

Clients SHOULD NOT immediately fetch the bundle and build a session as soon as a new device is announced. Before the first message is exchanged, the contact does not know which PreKey has been used (or, in fact, that any PreKey was used at all). As they have not had a chance to remove the used PreKey from their bundle announcement, this could lead to collisions where both Alice and Bob pick the same PreKey to build a session with a specific device. As each PreKey SHOULD only be used once, the party that sends their initial PreKeyWhisperMessage later loses this race condition. This means that they think they have a valid session with the contact, when in reality their messages MAY be ignored by the other end. By postponing building sessions, the chance of such issues occurring can be drastically reduced. It is RECOMMENDED to construct sessions only immediately before sending a message.

As there are no explicit error messages in this protocol, if a client does receive a PreKeyWhisperMessage using an invalid PreKey, they SHOULD respond with a KeyTransportElement, sent in a <*message*> using a PreKeyWhisperMessage. By building a new session with the original sender this way, the invalid session of the original sender will get overwritten with this newly created, valid session.

If a PreKeyWhisperMessage is received as part of a Message Archive Management (XEP-0313) catch-up and used to establish a new session with the sender, the client SHOULD postpone deletion of the private key corresponding to the used PreKey until after MAM catch-up is completed. If this is done, the client MUST then also send a KeyTransportMessage using a PreKeyWhisperMessage before sending any payloads using this session, to trigger re-keying. (as above) This practice can mitigate the previously mentioned race condition by preventing message loss.

As the asynchronous nature of OMEMO allows decryption at a later time to currently offline devices client SHOULD include a Message Processing Hints (XEP-0334) <*store* /> hint in their OMEMO messages. Otherwise, server implementations of Message Archive Management (XEP-0313) will generally not retain OMEMO messages, since they do not contain a <*body* />

6. Implementation Notes

For details on axoltol, see the specification and reference implementation.

The axolotl library's reference implementation (and presumably its ports to various other platforms) uses a trust model that doesn't work very well with OMEMO. For this reason it may be desirable to have the library consider all keys trusted, effectively disabling its trust management. This makes it necessary to implement trust handling oneself.

7. Security Considerations

Clients MUST NOT use a newly built session to transmit data without user intervention. If a client were to opportunistically start using sessions for sending without asking the user whether to trust a device first, an attacker could publish a fake device for this user, which would then receive copies of all messages sent by/to this user. A client MAY use such "not (yet) trusted" sessions for decryption of received messages, but in that case it SHOULD indicate the untrusted nature of such messages to the user.

When prompting the user for a trust decision regarding a key, the client SHOULD present the user with a fingerprint in the form of a hex string, QR code, or other unique representation, such that it can be compared by the user.

While it is RECOMMENDED that clients postpone private key deletion until after MAM catch-up and this standards mandates that clients MUST NOT use duplicate-PreKey sessions for sending, clients MAY delete such keys immediately for security reasons. For additional information on potential security impacts of this decision, refer to Menezes, Alfred, and Berkant Ustaoglu. "On reusing ephemeral keys in Diffie-Hellman key agreement protocols." International Journal of Applied Cryptography 2, no. 2 (2010): 154-158..

In order to be able to handle out-of-order messages, the axolotl stack has to cache the keys belonging to "skipped" messages that have not been seen yet. It is up to the implementor to decide how long and how many of such keys to keep around. 8. IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA).

9. XMPP Registrar Considerations

9.1 Protocol Namespaces

This specification defines the following XMPP namespaces:

urn:xmpp:omemo:0

The XMPP Registrar shall include the foregoing namespace in its registry at <https://xmpp.org/registrar/namespaces.html>, as goverened by XMPP Registrar Function (XEP-0053).

9.2 Protocol Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of **XEP-0053**.

10. XML Schema

Xml Schema

```
<xml version="1.0" encoding="utf8">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:xmpp:omemo:0"
    xmlns="urn:xmpp:omemo:0">

<xs:element name="encrypted">
    <xs:element name="header">
    <xs:attribute name="sid" type="xs:integer"/>
    <xs:complexType>
        <xs:sequence>
        <xs:element name="key" type="xs:base64Binary" maxOccurs="unbounded">
            <xs:attribute name="rid" type="xs:integer"/>
        </xs:element>
        <xs:element name="iv" type="xs:base64Binary"/>
    </xs:complexType>
    </xs:element>
    <xs:element name="payload" type="xs:base64Binary" minOccurs="0"/>
</xs:element>

<xs:element name="list">
    <xs:complexType>
    <xs:sequence>
        <xs:element name="device" maxOccurs="unbounded">
        <xs:attribute name="id" type="integer"/>
        </xs:element>
    </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="bundle">
    <xs:complexType>
    <xs:sequence>
        <xs:element name="signedPreKeyPublic" type="base64Binary">
        <xs:attribute name="id" type="integer"/>
        </xs:element>
        <xs:element name="signedPreKeySignature" type="base64Binary"/>
        <xs:element name="identityKey" type="base64Binary"/>
        <xs:element name="prekeys">
        <xs:complexType>
            <xs:sequence>
            <xs:element name="preKeyPublic" type="base64Binary" maxOccurs="unbounded">
                <xs:attribute name="id" type="integer"/>
            </xs:element>
            </xs:sequence>
        </xs:complexType>
        </xs:element>
    </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

11. Acknowledgements

Big thanks to Daniel Gultsch for mentoring me during the development of this protocol. Thanks to Thijs Alkemade and Cornelius Aschermann for talking through some of the finer points of the protocol with me. And lastly I would also like to thank Sam Whited, Holger Weiss, and Florian Schmaus for their input on the standard.

Appendices

Appendix A: Document Information

Series:: XEP

Number:: xxxx

Publisher:: XMPP Standards Foundation

Status:: ProtoXEP

Type:: Standards Track

Version:: 0.0.1

Last Updated:: 2015-10-25

Approving Body:: XMPP Council

Dependencies:: XMPP Core, XEP-0163

Supersedes:: None

Superseded By:: None

Short Name:: NOT_YET_ASSIGNED

This document in other formats:: XML PDF

Appendix B: Author Information

Andreas Straub

Email:: andy@strb.org

JabberID:: andy@strb.org

Appendix C: Legal Notices

Copyright

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Disclaimer of Warranty

Note: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. In no event shall the XMPP Standards Foundation or the authors of this Specification be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification.

Limitation of Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising out of the use or inability to use the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

IPR Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which may be found at <https://xmpp.org/extensions/ipr-policy.shtml> or obtained by writing to XSF, P.O. Box 1641, Denver, CO 80201 USA).

Appendix D: Relation to XMPP

The Extensible Messaging and Presence Protocol (XMPP) is defined in the XMPP Core (RFC 6120) and XMPP IM (RFC 6121) specifications contributed by the XMPP Standards Foundation to the Internet Standards Process, which is managed by the Internet Engineering Task Force in accordance with RFC 2026. Any protocol defined in this document has been developed outside the Internet Standards Process and is to be understood as an extension to XMPP rather than as an evolution, development, or modification of XMPP itself.

Appendix E: Discussion Venue

The primary venue for discussion of XMPP Extension Protocols is the <standards@xmpp.org> discussion list.

Discussion on other xmpp.org discussion lists might also be appropriate; see <https://xmpp.org/about/discuss.shtml> for a complete list.

Errata can be sent to <editor@xmpp.org>.

Appendix F: Requirements Conformance

The following requirements keywords as used in this document are to be interpreted as described in RFC 2119: “MUST”, “SHALL”, “REQUIRED”; “MUST NOT”, “SHALL NOT”; “SHOULD”, “RECOMMENDED”; “SHOULD NOT”, “NOT RECOMMENDED”; “MAY”, “OPTIONAL”.

Appendix G: Notes

1. XEP-0364: Current Off-the-Record Messaging Usage <https://xmpp.org/extensions/xep-0364.html>.
2. XEP-0027: Current Jabber OpenPGP Usage <https://xmpp.org/extensions/xep-0027.html>.
3. XEP-0280: Message Carbons <https://xmpp.org/extensions/xep-0280.html>.
4. XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
5. XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.
6. XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
7. XEP-0334: Message Processing Hints <https://xmpp.org/extensions/xep-0334.html>.
8. XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
9. Menezes, Alfred, and Berkant Ustaoglu. “On reusing ephemeral keys in Diffie-Hellman key agreement protocols.” International Journal of Applied Cryptography 2, no. 2 (2010): 154-158.
10. The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

11. XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>. Appendix H: Revision History

Note: Older versions of this specification might be available at https://xmpp.org/extensions/attic/ Version 0.0.1 (2015-10-25)

First draft. (as)

CHAPTER 5

Reference

OmemoState

class omemo.state.**OmemoState**(*own_jid*, *connection*)

__init__(*own_jid*, *connection*)
Instantiates an OmemoState object.

Parameters **connection** – an sqlite3.Connection

__module__ = 'omemo.state'

add_device(*name*, *device_id*)

add_own_device(*device_id*)

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

bundle
.. *highlight* – python Returns all data needed to announce bundle information.

```
bundle_dict = {
    'signedPreKeyPublic': bytes,
    'prekeys': [(int, bytes) (int, bytes)],
    'identityKey': bytes,
    'signedPreKeyId': int,
    'signedPreKeySignature': bytes
}
```

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

decrypt_msg(*msg_dict*)

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

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.set_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

set_devices (*name*, *devices*)
Return a an.

Parameters

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

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

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

CHAPTER 6

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 7

Authors

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

CHAPTER 8

Changelog

0.1.0 (2016-01-11)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

D

G

H

O

S