# py-wsse Documentation

***Release 0.1***

**Carl Meyer**

February 14, 2017

# Contents

1 Prerequisites 3

2 Installation 5

3 Features 7

4 Usage 9
4.1 With Suds . . . . . . 9
4.2 Standalone functions . . . . . . 10

5 Contributing 11

WS-Security (SOAP WSSE) support for Python, including an optional Suds plugin.

# Prerequisites

`py-wsse` supports Python 2.7, 3.3, and 3.4.

`py-wsse` depends on PyOpenSSL, python-xmlsec, and lxml, which in turn rely on C headers being available on your system for `OpenSSL`, `libxml2`, and `libxmlsec1`. On Debian/Ubuntu, `sudo apt-get install libssl-dev libxml2-dev libxmlsec1-dev` should take care of that. On RedHat-based systems, try `sudo yum install openssl-devel libxml2-devel xmlsec1-devel xmlsec1-openssl-devel libtool-ltdl-devel`.

If using Suds, the jurko fork is required; it contains required fixes to the plugin API. (This fork is available on PyPI as the suds-jurko package.)

CHAPTER 2

# Installation

`py-wsse` is available on PyPI. Install it with:

```
pip install py-wsse
```

Or use `pip install py-wsse[suds]` to pull in Suds as an additional dependency.

CHAPTER 3

# Features

`py-wsse` supports exactly what I needed and no more. If you need more, or more flexibility, pull requests with tests and doc updates are welcome! Current features:

- Signing a SOAP envelope `Body` and `wsu:Timestamp` security token using an X509 certificate and associated private key.
- Verifying WSSE signatures on a received SOAP envelope.
- Encrypting the contents of the SOAP `Body` using the recipient's X509 certificate.
- Decrypting `EncryptedData` elements in a received SOAP envelope.

**Warning:** Yes, XML Encryption 1.0 is broken. Sometimes people use it anyway – require it, even – and we have to talk to their systems, so here it is. Just don't assume it will actually protect your data. Use TLS.

# CHAPTER 4

# Usage

## 4.1 With Suds

To use with Suds, just add an instance of `wsse.suds.WssePlugin` to the list of `plugins` passed to a new `Client` instance:

```python
from suds.client import Client
from suds.wsse import Security, Timestamp
from wsse.suds import WssePlugin

def get_client(our_keyfile_path, our_certfile_path, their_certfile_path):
    wsse = Security()
    wsse.tokens.append(Timestamp())

    return Client(
        wsdl_url,
        transport=transport,
        wsse=wsse,
        plugins=[
            WssePlugin(
                keyfile=our_keyfile_path,
                certfile=our_certfile_path,
                their_certfile=their_certfile_path,
            ),
        ],
    )
```

`WssePlugin` requires that the outgoing messages already have a `wsse:Security` element in the `soap:Header` with a `wsu:Timestamp` token. Suds can do this via its `Security` and `Timestamp` objects, as shown in the above example.

In the example, `our_keyfile_path`, `our_certfile_path`, and `their_certfile_path` should all be absolute filesystem paths to X509 certificates (or private key) in PEM format. The `our` cert and key are used to sign outgoing messages and decrypt incoming messages. The `their` cert is used to encrypt outgoing messages and verify the signature on incoming messages.

Note that `WssePlugin` is currently hardcoded to sign the `wsu:Timestamp` and `soap:Body` elements, and to encrypt only the first child of the `soap:Body` element. Pull requests to add more flexibility are welcome.

## 4.2 Standalone functions

If you aren't using Suds you can also use the encryption/decryption and signing/verification capabilities of `py-wsse` directly. The functions are `wsse.signing.sign`, `wsse.signing.verify`, `wsse.encryption.encrypt`, and `wsse.encryption.decrypt`. For usage details, see how they are used by `wsse.suds.SigningPlugin` (in which `context.envelope` and `context.reply` are strings containing XML documents) and/or see their respective docstrings.

CHAPTER 5

# Contributing

See the contributing docs.