pyoauth Documentation

Release 0.0.1

Yesudeep Mangalapilly

November 02, 2014

Contents

A Python library that implements the OAuth protocol for clients and servers.

CHAPTER 1

Getting the library

```
$ pip install pyoauth
```

or

```
$ git clone git://github.com/gorakhargosh/pyoauth.git
```

or

```
$ git clone http://code.google.com/p/pyoauth/
```

```
$ cd pyoauth
$ python setup.py install
```

CHAPTER 2

About the implementation

PyOAuth implements versions 1.0 and 2.0 of the OAuth protocol as per the RFC specifications (See RFC5849), which supersede any previous versions of the protocol.

Client classes do not send HTTP requests but implement enough of the OAuth protocol to help you build request proxies that can be used to send actual HTTP requests. In essence, it implements OAuth and nothing else. This is a very conscious decision by the library authors. It allows framework authors and API users to use the library without pulling in unnecessary dependencies which may not work on their platform of choice. For example, you can use any of httplib2, tornado, webapp2, or django to send HTTP requests built with this library.

Wherever possible the implementation tries to warn you about problems you may encounter when processing or building OAuth requests by using a fail-fast approach—we try to tell you as much about the problem as possible. For example, OAuth relies on the availability of SSL to communicate securely, and therefore, the library checks whether the OAuth endpoint URLs you specify use SSL and prohibits you from using them if they do not begin with `https://`. You can change this behavior to suit your needs, but the library will warn you.

Custom HTTP methods are currently not supported.

2.1 Signature methods

All the signature methods recommended by the OAuth specification have been implemented by this library, namely:

1. PLAINTEXT
2. HMAC-SHA1
3. RSA-SHA1

Custom signature methods are currently not supported.

2.1.1 RSA-SHA1 requirements

The RSA-SHA1 signature method relies on the availability of third-party libraries like pyasn1, and PyCrypto or M2Crypto.

The library accepts PEM-encoded X.509 certificates, RSA public keys, and RSA private keys. The validity of the X.509 certificates will not be verified by the signing functions. You must ensure the validity of certificates when you accept them by using other utility methods provided by this library or any other suitable means. It is also a good idea to periodically remind your clients about their certificate expiration dates—human beings forget after all.

For a quick rundown about these certificates and keys, please read *Using RSA-SHA1 signatures*.

CHAPTER 3

User Guides

3.1 A typical OAuth 1.0 flow in simple words

1. Construct a client with its client credentials.
2. Send an HTTPS request for temporary credentials with a callback URL which the server will call with an OAuth verification code after authorizing the resource owner (end-user).
3. Obtain temporary credentials from a successful server response.
4. Use the temporary credentials to build an authorization URL and redirect the resource owner (end-user) to the generated URL.
5. If a callback URL is not provided when requesting temporary credentials, the server displays the OAuth verification code to the resource owner (end-user), which she then types into your application.

   OR

   If a callback URL is provided, the server redirects the resource owner (end-user) after authorization to your callback URL attaching the OAuth verification code as a query parameter.
6. Using the obtained OAuth verification code from step 5 and the temporary credentials obtained in step 3, send an HTTPS request for token credentials.
7. Obtain token credentials from a successful server response.
8. Save the token credentials for future use (say, in a database).

3.1.1 Accessing a resource

1. Construct a client with its client credentials.
2. Using the token credentials that you have saved (say, in a database), send an HTTP request to a resource URL.
3. Obtain the response and deal with it.

3.2 Using RSA-SHA1 signatures

```
-----BEGIN RSA PRIVATE KEY-----
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBALRiMLAh9iimur8V
A7qVvdqxevEuUkW4K+2KdMXmnQbG9Aa7k7eBjK1S+0LYmVjPKlJGNXHDGuy5Fw/d
7rjVJ0BLB+ubPK8iA/Tw3hLQgXMRRGRXXCn8ikfuQfjUS1uZSatdLB81mydBETlJ
hI6GH4twrbDJCR2Bwy/XWXgqgGRzAgMBAAECgYBYWVtleUzavkbrPjy0T5FMou8H
```

```
X9u2AC2ry8vD/l7cqedtwMPp9k7TubgNFo+NGvKsl2ynyprOZR1xjQ7WgrgVB+mm
uScOM/5HVceFuGRDhYTCObE+y1kxRloNYXnx3ei1zbeYLPCHdhxRYW7T0qcynNmw
rn05/KO2RLjgQNalsQJBANeA3Q4Nugqy4QBUCEC09SqylT2K9FrrItqL2QKc9v0Z
zO2uwllCbg0dwpVuYPYXYvikNHHg+aCWF+VXsb9rpPsCQQDWR9TT4ORdzoj+Nccn
qkMsDmzt0EfNaAOwHOmVJ2RVBspPcxt5iN4HI7HNeG6U5YsFBb+/GZbgfBT3kpNG
WPTpAkBI+gFhjfJvRw38n3g/+UeAkwMI2TJQS4n8+hid0uus3/zOjDySH3XHCUno
cn1xOJAyZODBo47E+67R4jV1/gzbAkEAklJaspRPXP877NssM5nAZMU0/O/NGCZ+
3jPgDUno6WbJn5cqm8MqWhW1xGkImgRk+fkDBquiq4gPiT898jusgQJAd5Zrr6Q8
AO/0isr/3aa6O6NLQxISLKcPDk2NOccAfS/xOtfOz4sJYM3+Bs4Io9+dZGSDCA54
Lw03eHTNQghS0A==
-----END RSA PRIVATE KEY-----

-----BEGIN PUBLIC KEY-----
 MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0YjCwIfYoprq/FQO6lb3asXrx
 LlJFuCvtinTF5p0GxvQGu5O3gYytUvtC2JlYzypSRjVxwxrsuRcP3e641SdASwfr
 mzyvIgP08N4S0IFzEURkV1wp/IpH7kH41EtbmUmrXSwfNZsnQRE5SYSOhh+LcK2w
 yQkdgcMv11l4KoBkcwIDAQAB
 -----END PUBLIC KEY-----

-----BEGIN CERTIFICATE-----
MIIBpjCCAQ+gAwIBAgIBATANBgkqhkiG9w0BAQUFADAZMRcwFQYDVQQDDA5UZXN0
IFByaW5jaXBhbDAeFw03MDAxMDEwODAwMDBaFw0zODEyMzEwODAwMDBaMBkxFzAV
BgNVBAMMDlRlc3QgUHJpbmNpcGFsMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQC0YjCwIfYoprq/FQO6lb3asXrxLlJFuCvtinTF5p0GxvQGu5O3gYytUvtC2JlY
zypSRjVxwxrsuRcP3e641SdASwfrmzyvIgP08N4S0IFzEURkV1wp/IpH7kH41Etb
mUmrXSwfNZsnQRE5SYSOhh+LcK2wyQkdgcMv11l4KoBkcwIDAQABMA0GCSqGSIb3
DQEBBQUAA4GBAGZLPEuJ5SiJ2ryq+CmEGOXfvlTtEL2nuGtr9PewxkgnOjZpUy+d
4TvuXJbNQc8f4AMWL/tO9w0Fk80rWKp9ea8/df4qMq5qlFWlx6yOLQxumNOmECKb
WpkUQDIDJEoFUzKMVuJf4KO/FJ345+BNLGgbJ6WujreoM1X/gYfdnJ/J
-----END CERTIFICATE-----
```

3.2.1 Whoa?!

Betchyoo weren't expecting that. Ha!

"What's up with all that gibberish up there, man?" you ask.

```
  ________________________________________
 /                                        \
|                                          |
|   Yeah, dude. I mean, up there! -^       |
|                                          |
 \_________________________________________/
                                V
                               You. =O
```

That, my friend, is three friends. Wait, my grammar has left with the wind.

"Three friends, huh?"

Yep, three friends. They keep your privates private and your publics... err, they're not really worried about your publics.

"Do these friends have names, man?"

Yeah. They're:

1. RSA private key
2. RSA public key

3. X.509 certificate

They're a happy couple.

> "One. Two. Three. Hmm. Oye, that's 3 of them. Couple? So, umm... how do they work? Somethin' to do with opensll yeah?"

We'll get to this in a bit. Kinda, yes, and that's "OpenSSL," by the way.

3.2.2 Why should you use public-key encryption?

> Because it's harder.

No, I mean it's harder for you to work with. And it also allows attackers to snoop into everything you're saying *after they're long dead and gone*.

Consider a series of love letters Mr. Bunny sends to Mrs. Bunny. But the postman, Mr. Evil Fox, however, is a very cunning fox. He doesn't like Mr. Bunny, so he opens all of Mr. Bunny's letters and changes a few words before delivering them to trick Mrs. Bunny into thinking that Mr. Bunny is cheating on her.

> What Mr. Bunny sends:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|                                                          |
|   Dear Mrs. Bunny,                                       |
|                                                          |
|   I love you.                                            |
|                                                          |
|   Love, Mr. Bunny                                        |
|                                                          |
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

> What Mrs. Bunny receives:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|                                                          |
|   Dear Mrs. Hippopotamus,                                |
|                                                          |
|   I love you.                                            |
|                                                          |
|   Love, Mr. Bunny                                        |
|                                                          |
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

> All thanks to Mr. Evil Fox.

```
  ____________________________________
 /                                    \
|                                      |
|   Grrrrrrrr. >=|B                    |
|                                      |
 \____________________________________/
                                  V
                              Mrs. Bunny
```

Mrs. Bunny is furious and scolds Mr. Bunny about all the letters Mr. Bunny never sent! Mr. Bunny is perplexed and rubs Aladdin's lamp to ask the genie for a magical lock.

> A magical lock with two keys: one to lock it and the other to open it.

He gives the second key to Mrs. Bunny and only she can now open the envelopes that Mr. Bunny sends to her. Poor Mr. Evil Fox tries his best to open Mr. Bunny's letters, but can't anymore. Hooray!

What Mr. Bunny sends:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|                                                          |
|   Dear Mrs. Bunny,                                       |
|                                                          |
|   I love you. Hooray!                                    |
|                                                          |
|   Love, Mr. Bunny                                        |
|                                                          |
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

What Mrs. Bunny now receives:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|                                                          |
|   Dear Mrs. Bunny,                                       |
|                                                          |
|   I love you. Hooray!                                    |
|                                                          |
|   Love, Mr. Bunny                                        |
|                                                          |
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
Mr. Fox               ->  ='(
Mr. and Mrs. Bunny    ->  =B and =B
```

Imagine public-key encryption to be the equivalent of having a lock with two complementary keys—that is, if you lock something with one key, only the other key can open it. So, RSA public-key encryption uses the notion of these two keys to secure your messages in a way that:

1. Ensures the messages that A and B exchange are **confidential**.
2. Ensures the message that B receives from A were **actually sent by A** and vice versa.
3. Ensures the messages that A and B exchange **aren't tampered with while in transit**.

Now, using RSA-SHA-1 signatures shouldn't be as difficult to use as they are with OAuth. Therefore, we have tried to take the burden of implementing them away from you and made it as easy as possible for you to use this signature method.

So how does OAuth use public-key encryption?

OAuth requires the use of SSL by clients when requesting token secrets from OAuth servers. Verifying the authenticity of the messages is handled by RSA-SHA-1 signatures.

Here is what you have to do to use your RSA key-pair with OAuth:

1. You **share your public key** (an RSA public key or an X.509 public-key certificate) **with the OAuth provider**.
2. Sign your messages with your RSA private key (which you keep safe and don't share with anybody else including the OAuth provider) by telling the request building methods to **use `"RSA-SHA1"` as the signature method** and your **RSA private key as the client secret**. Easy, huh?

The OAuth provider can now use your public key to verify the messages that you send to it after you sign them with your private key.

That's essentially it.

3.2.3 What does a key look like?

A key can come in multiple formats. The PEM, or Privacy Enhanced electronic-Mail, format is a commonly accepted format, and that is the one preferred by this library. Keys can be also stored in JSON formats as the one used by the Keyczar library. Have a look at the Keyczar RSA private key format. Public keys that you generate are generally encoded into something called an X.509 public-key certificate. You can share either an RSA public key or an X.509 public-key certificate with your OAuth provider. Providers usually ask for an X.509 public-key certificate.

RSA Private Key

An example:

```
-----BEGIN RSA PRIVATE KEY-----
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBALRiMLAh9iimur8V
A7qVvdqxevEuUkW4K+2KdMXmnQbG9Aa7k7eBjK1S+0LYmVjPKlJGNXHDGuy5Fw/d
7rjVJ0BLB+ubPK8iA/Tw3hLQgXMRRGRXXCn8ikfuQfjUS1uZSatdLB81mydBETlJ
hI6GH4twrbDJCR2Bwy/XWXgqgGRzAgMBAAECgYBYWVtleUzavkbrPjy0T5FMou8H
X9u2AC2ry8vD/l7cqedtwMPp9k7TubgNFo+NGvKsl2ynyprOZR1xjQ7WgrgVB+mm
uScOM/5HVceFuGRDhYTCObE+y1kxRloNYXnx3ei1zbeYLPCHdhxRYW7T0qcynNmw
rn05/KO2RLjgQNalsQJBANeA3Q4Nugqy4QBUCEC09SqylT2K9FrrItqL2QKc9v0Z
zO2uwllCbg0dwpVuYPYXYvikNHHg+aCWF+VXsb9rpPsCQQDWR9TT4ORdzoj+Nccn
qkMsDmzt0EfNaAOwHOmVJ2RVBspPcxt5iN4HI7HNeG6U5YsFBb+/GZbgfBT3kpNG
WPTpAkBI+gFhjfJvRw38n3g/+UeAkwMI2TJQS4n8+hid0uus3/zOjDySH3XHCUno
cn1xOJAyZODBo47E+67R4jV1/gzbAkEAklJaspRPXP877NssM5nAZMU0/O/NGCZ+
3jPgDUno6WbJn5cqm8MqWhW1xGkImgRk+fkDBquiq4gPiT898jusgQJAd5Zrr6Q8
AO/0isr/3aa6O6NLQxISLKcPDk2NOccAfS/xOtfOz4sJYM3+Bs4Io9+dZGSDCA54
Lw03eHTNQghS0A==
-----END RSA PRIVATE KEY-----
```

RSA Public Key

An example:

```
-----BEGIN PUBLIC KEY-----
 MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0YjCwIfYoprq/FQO6lb3asXrx
 LlJFuCvtinTF5p0GxvQGu5O3gYytUvtC2JlYzypSRjVxwxrsuRcP3e641SdASwfr
 mzyvIgP08N4S0IFzEURkV1wp/IpH7kH41EtbmUmrXSwfNZsnQRE5SYSOhh+LcK2w
 yQkdgcMv11l4KoBkcwIDAQAB
 -----END PUBLIC KEY-----
```

X.509 Public-key Certificate

An example:

```
-----BEGIN CERTIFICATE-----
MIIBpjCCAQ+gAwIBAgIBATANBgkqhkiG9w0BAQUFADAZMRcwFQYDVQQDDA5UZXN0
IFByaW5jaXBhbDAeFw03MDAxMDEwODAwMDBaFw0zODEyMzEwODAwMDBaMBkxFzAV
BgNVBAMMDlRlc3QgUHJpbmNpcGFsMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQC0YjCwIfYoprq/FQO6lb3asXrxLlJFuCvtinTF5p0GxvQGu5O3gYytUvtC2JlY
zypSRjVxwxrsuRcP3e641SdASwfrmzyvIgP08N4S0IFzEURkV1wp/IpH7kH41Etb
mUmrXSwfNZsnQRE5SYSOhh+LcK2wyQkdgcMv11l4KoBkcwIDAQABMA0GCSqGSIb3
DQEBBQUAA4GBAGZLPEuJ5SiJ2ryq+CmEGOXfvlTtEL2nuGtr9PewxkgnOjZpUy+d
4TvuXJbNQc8f4AMWL/tO9w0Fk80rWKp9ea8/df4qMq5qlFWlx6yOLQxumNOmECKb
WpkUQDIDJEoFUzKMVuJf4KO/FJ345+BNLGgbJ6WujreoM1X/gYfdnJ/J
-----END CERTIFICATE-----
```

3.2.4 Making your own key-pair

"Alright. I get it Sherlock. Now, how do I make my pair of keys to use with an OAuth provider?"

You can generate your own self-signed X.509 certificate and private RSA key using the tools OpenSSL provides. You'll need OpenSSL installed for the following command to work at the terminal:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -sha1 -keyout \
    rsa_private_key.pem -out x509_public_certificate.pem
```

Answer all the questions that the tool asks and you should be good to go. For more detailed information about generating X.509 public-key certificates, read:

1. http://www.ipsec-howto.org/x595.html
2. http://www.imacat.idv.tw/tech/sslcerts.html#reqform
3. http://code.google.com/apis/gdata/docs/auth/oauth.html#openssl

3.3 Contributing

Welcome hackeratti! So you have got something you would like to see in `pyoauth`? Whee. This document will help you get started.

3.3.1 Important URLs

`pyoauth` uses git to track code history and hosts its code repository at github. The issue tracker is where you can file bug reports and request features or enhancements to `pyoauth`.

3.3.2 OAuth Test server information for testing clients

Sandbox URL: `http://oauth-sandbox.sevengoslings.net/` Username: `pyoauth` Password (Guard Kitten): `Kitty-Agent "Able"`

URLs

Request token URL: `http://oauth-sandbox.sevengoslings.net/request_token` User authorization URL: `http://oauth-sandbox.sevengoslings.net/authorize` Access token URL: `http://oauth-sandbox.sevengoslings.net/access_token`

Two-legged resource URL: `http://oauth-sandbox.sevengoslings.net/two_legged` Three-legged resource URL: `http://oauth-sandbox.sevengoslings.net/three_legged`

Consumer keys:

Consumer key: `ac19e45c6b01a767` Consumer secret: `59806917a29a94ee77190ec06c50`

Nonce checking is enabled.

3.3.3 Before you start

Ensure your system has the following programs and libraries installed before beginning to hack:

1. Python
2. git
3. ssh

3.3.4 Setting up the Work Environment

`pyoauth` makes extensive use of zc.buildout to set up its work environment. You should get familiar with it.

Steps to setting up a clean environment:

1. Fork the code repository into your github account. Let us call you `hackeratti`. That *is* your name innit? Replace `hackeratti` with your own username below if it isn't.
2. Clone your fork and setup your environment:

```
$ git clone --recursive git@github.com:hackeratti/pyoauth.git
$ cd pyoauth
$ python bootstrap.py --distribute
$ bin/buildout
```

Important: Re-run `bin/buildout` every time you make a change to the `buildout.cfg` file.

That's it with the setup. Now you're ready to hack on `pyoauth`.

3.3.5 Enabling Continuous Integration

The repository checkout contains a script called `autobuild.sh` which you should run prior to making changes. It will detect changes to Python source code or restructuredText documentation files anywhere in the directory tree and rebuild sphinx documentation, run all tests using nose, and generate coverage reports.

Start it by issuing this command in the `pyoauth` directory checked out earlier:

```
$ tools/autobuild.sh
...
```

Happy hacking!

CHAPTER 4

API Documentation

4.1 Client implementations

4.1.1 *pyoauth.oauth1.protocol*

4.1.2 *pyoauth.oauth1.client*

4.1.3 *pyoauth.oauth1.client.google*

4.2 OAuth 1.0-specific utilities

4.2.1 *pyoauth.oauth1*

4.3 Utilities

4.3.1 *pyoauth.url*

4.3.2 *pyoauth.http*

CHAPTER 5

Contribute

Found a bug in or want a feature added to pyoauth? You can fork the official code repository or file an issue ticket at the issue tracker. You may also want to refer to *Contributing* for information about contributing code or documentation to pyoauth.

5.1 Contributing

Welcome hackeratti! So you have got something you would like to see in pyoauth? Whee. This document will help you get started.

5.1.1 Important URLs

pyoauth uses git to track code history and hosts its code repository at github. The issue tracker is where you can file bug reports and request features or enhancements to pyoauth.

5.1.2 OAuth Test server information for testing clients

Sandbox URL: `http://oauth-sandbox.sevengoslings.net/` Username: `pyoauth` Password (Guard Kitten): `Kitty-Agent "Able"`

URLs

Request token URL: `http://oauth-sandbox.sevengoslings.net/request_token` User authorization URL: `http://oauth-sandbox.sevengoslings.net/authorize` Access token URL: `http://oauth-sandbox.sevengoslings.net/access_token`

Two-legged resource URL: `http://oauth-sandbox.sevengoslings.net/two_legged` Three-legged resource URL: `http://oauth-sandbox.sevengoslings.net/three_legged`

Consumer keys:

Consumer key: `ac19e45c6b01a767` Consumer secret: `59806917a29a94ee77190ec06c50`

Nonce checking is enabled.

5.1.3 Before you start

Ensure your system has the following programs and libraries installed before beginning to hack:

1. Python
2. git
3. ssh

5.1.4 Setting up the Work Environment

`pyoauth` makes extensive use of zc.buildout to set up its work environment. You should get familiar with it.

Steps to setting up a clean environment:

1. Fork the code repository into your github account. Let us call you `hackeratti`. That *is* your name innit? Replace `hackeratti` with your own username below if it isn't.
2. Clone your fork and setup your environment:

```
$ git clone --recursive git@github.com:hackeratti/pyoauth.git
$ cd pyoauth
$ python bootstrap.py --distribute
$ bin/buildout
```

Important: Re-run `bin/buildout` every time you make a change to the `buildout.cfg` file.

That's it with the setup. Now you're ready to hack on `pyoauth`.

5.1.5 Enabling Continuous Integration

The repository checkout contains a script called `autobuild.sh` which you should run prior to making changes. It will detect changes to Python source code or restructuredText documentation files anywhere in the directory tree and rebuild sphinx documentation, run all tests using nose, and generate coverage reports.

Start it by issuing this command in the `pyoauth` directory checked out earlier:

```
$ tools/autobuild.sh
...
```

Happy hacking!

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*