gpglib

Release 0.1

April 29, 2015

Contents

This library was made with a particular need in mind and hence does the bare minimum to achieve that:

- Parse PGP RSA Secret Keys
- Decrypt CAST5 encrypted, ZIP compressed PGP messages

It provides a simple interface to do this with (see *Example*) and done with a readable/understandable implementation.

CHAPTER 1

Some Background

This library was created out of frustration with how slow the python libraries for parsing PGP messages were.

As it turns out, all the other libraries (except one) do their work by shelling out to the `gpg` binary. Our requirements were to process a large number of small PGP messages and ideally without batching them. With the existing libraries we were only getting around 20 messages a second due to the overhead of shelling out to gpg.

The one library we found that didn't shell out was a magnificent thing called OpenPGP, which can be found over at http://pypi.python.org/pypi/OpenPGP. Unfortunately this library was last edited 7 years ago and is about as slow as shelling out.

We decided that we could do better and so started our own RFC4880 compliant PGP parser.

Some initial tests show that gpglib can get around 300 messages a second (when pycrypto is compiled with fast math).

CHAPTER 2

References

We mainly used the following references to make this parser:

- RFC4880 (http://tools.ietf.org/html/rfc4880)
- OpenPGP (http://pypi.python.org/pypi/OpenPGP)
- OpenPGP SDK (http://openpgp.nominet.org.uk/cgi-bin/trac.cgi)
- Python pgpdump (http://pypi.python.org/pypi/pgpdump/1.3)
- C pgpdump (http://www.mew.org/~kazu/proj/pgpdump/en/)
- libsimplepgp (https://github.com/mrmekon/libsimplepgp)

CHAPTER 3

Libraries

This library isn't possible without:

- Pycrypto (https://www.dlitz.net/software/pycrypto/)
- Bitstring (http://packages.python.org/bitstring/)

CHAPTER 4

Example

```
from gpglib.structures import EncryptedMessage, Key

data = open('tests/data/key.secret.gpg').read()
key = Key(passphrase='blahandstuff')
key.parse(data)
keys = key.key_dict()
print keys

data = open('tests/data/data.small.dump.gpg').read()
message = EncryptedMessage(keys)
message.decrypt(data)

print "Message successfully decrypted data.dump::"
print message.plaintext

data = open('tests/data/data.big.dump.gpg').read()
message = EncryptedMessage(keys)
message.decrypt(data)

print "Message successfully decrypted data.big.dump::"
print message.plaintext
```

CHAPTER 5

GPG Lib

We couldn't find a library for decrypting gpg that didn't shell out to gpg.

And shelling out to gpg is slow when you do it for many small files.

So, with the help of http://pypi.python.org/pypi/OpenPGP and PyCrypto we created this, which is more performant than shelling out....

CHAPTER 6

Installing

To install, just use pip:

```
$ pip install gpglib
```

Or download from pypi: http://pypi.python.org/pypi/gpglib.

Or clone the git repo: https://github.com/Hitwise/gpglib.

CHAPTER 7

Making test data

This is what I did to get the data in tests/data.

From within tests/data:

```
$ gpg --gen-key --homedir ./gpg
# Once with RSA encrypt and sign, username Stephen and password "blahandstuff"
# And again with DSA/Elgamal, username Bobby and password "blahandstuff"
```

Then find the keyid:

```
$ gpg --homedir ./gpg --list-keys
    #     ./gpg/pubring.gpg
    # -----------------
    # pub   2048R/1E42B68C 2012-06-15
    # uid                  Stephen
    # sub   2048R/80C7020A 2012-06-15
# Here, the key we want is "80C7020A"
```

Then with that keyid export the secret and public keys for both the rsa and dsa keys:

> $ gpg –export 80C7020A > key.public.rsa.gpg $ gpg –export-secret-key 80C7020A > key.secret.rsa.gpg

I then created dump.small and dump.big as random json structures (the big on is from http://json.org/example.html).

Then used the following command to populate the tests/data/encrypted folder:

> $ gpg -o encrypted/<key>/<cipher>/<compression>/<msg>.gpg –cipher-algo <cipher> –compress-algo <compression> –yes –disable-mdc –homedir ./gpg -r <name for key> –encrypt dump.<msg>

Where:

- <key> is rsa or dsa
- <cipher> is cast5, aes, 3des or blowfish
- <compression> is zip, zlib or bzip2
- <msg> is small and big for the two examples I have

CHAPTER 8

Tests

Install the pip requirements:

```
$ pip install -r requirements_test.txt
```

Install nosy if you want the ability to make tests autorun when you run the tests (https://bitbucket.org/delfick/nosy)

And then run:

```
$ ./test.sh
```

Or if you have nosy:

```
$ nosy ./test.sh
```

Currently not much is tested.

CHAPTER 9

Docs

Install the pip requirements:

```
$ pip install -r requirements_docs.txt
```

And then go into the docs directory and run make:

```
$ cd docs
$ make html
```

Open up docs/_build/html/index.html in your browser.

Automatically generated documentation is available at: http://gpglib.readthedocs.org/en/latest/