A proposal for a common OpenPGP test suite

Justus Winter <justus@gnupg.org>

2016-09-09

Testing GnuPG

- Reworked GnuPG's OpenPGP test suite (tests/openpgp)
- As of GnuPG 2.1.14, those tests are written in Scheme
  - portability
  - code reuse

Typical test case

```
(for-each-p
 "Checking encryption"
 (lambda (source)
   (tr:do
    (tr:open source)
    (tr:gpg "" `(--yes --encrypt --recipient ,usrname2))
    (tr:gpg "" '(--yes))
    (tr:assert-identity source)))
 (append plain-files data-files))
```

Shameless plug: The Python bindings for GPGME

- Will ship with GPGME 1.7
- Focus on ease of use, pythonicity, discoverability
- Comes with an extensive test suite
- Can be built out of tree, see 'pyme3' on PyPI

pyme3 hello world

```
import pyme
with pyme.Context(armor=True) as c:
    ciphertext, _, _ = c.encrypt(b"Hello Python world :)")
```

Testing GnuPG and friends

- Distinct test suites for
  - GnuPG
    - classic
    - modern
    - different versions...
  - GPGME
    - C
    - C++
    - Python
    - (CommonLisp)
    - ...
  - TinyGPG
- Rather obvious idea: merge the test suites

How do we get there

- Define an interface
  - "gpg-the-binary"
  - GPGME
  - new interface
- Port tests, write new ones
- Create test vectors
  - Challenge: randomness
  - decrypt(encrypt(text)) == text
  - cipher := encrypt(text), ship cipher, check decrypt(cipher) == text
- Standalone project
- Ship it like software

Benefits

- For us:
  - less maintenance work
  - increased test coverage
- For other OpenPGP implementations:
  - free test suite
  - measure features and compatibility
- For users:
  - better software
  - increased compatibility
  - test in production-like environments

Talk to me!

Test nerd? Developing an OpenPGP implementation? Interested?

Bonus slide: Why Scheme? Why not X...?

- Needed tests to be as portable as GnuPG (Windows!)
- TinySCHEME, ANSI C interpreter in like three files
- Using GnuPG's own platform abstractions
- Official GNU extension language
- Previous tests written in Bourne Shell
  - + everyone is somewhat familiar with it
  - + good at sequential execution of programs, pipes
  - - not a nice language
  - - hard to write portable scripts
  - - doesn't work on Windows
- Scheme tests
  - + expressive language
  - o transformation monad, pipe monad
  - - people are less familiar with Scheme
  - - debuggability, error messages