Monkeysign Documentation

Release ???

Antoine Beaupré

Jun 06, 2017

Contents

Monkeysign is a tool to overhaul the OpenPGP keysigning experience and bring it closer to something that most primates can understand.

The project makes use of cheap digital cameras and the type of bar code known as a *QRcode* to provide a human-friendly yet still-secure keysigning experience.

No more reciting tedious strings of hexadecimal characters. And, you can build a little rogue's gallery of the people that you have met and exchanged keys with! (Well, not yet, but it's part of the plan.)

Monkeysign also features a user-friendly commandline tool, similar to `caff`, to sign OpenPGP keys following the current best practices.

Monkeysign was written by Jerome Charaoui and Antoine Beaupre and is licensed under GPLv3.

CHAPTER 1

Features

- commandline and GUI interface
- GUI supports exchanging fingerprints with qrcodes
- print your OpenPGP fingerprint on a QRcode
- key signature done on a separate keyring
- signature sent in an encrypted email to ensure:

1. the signee controls the signed email
2. the signee controls the private key
3. the signee decides what to do with the signature

- local (“non-exportable”) signatures
- send through local email server, arbitrary SMTP server or other programs

For usage instructions, see *Usage* section, for install instructions, see *Install* section and for support, see the *Contribute* section.

CHAPTER 2

Similar projects

- OpenKeychain, a fork of APG, has support for exporting and importing fingerprints in QRcode and NFC. It uses similar strings for QRcodes exchanges and is compatible with Monkeysign. (Github project)
- GPG for Android (of the Guardian project) will import public keys in your device's keyring when they are found in QRcodes, so it should be able to talk with Monkeysign, but this remains to be tested. (Github project)
- Gibberbot (also of the Guardian project) can exchange OTR fingerprints using QRcodes. (Github project)

Install

Monkeysign can be installed in various ways, depending on which platform you are using. You can install Monkeysign:

- *Using packages*, recommended if you are running a distribution that has native packages for Monkeysign
- *Using PIP*, recommended if you are on another distribution that doesn't have native packages of Monkeysign or you want to run the latest version without upgrading the whole operating system
- *From source*, if the above doesn't work, if you need to test unreleased code, or if you want to contribute to Monkeysign

Using packages

Some distributions offer ready-to-use packages for Monkeysign which can be easily installed with a package manager. Below is a table of distributions that have packages for Monkeysign.

Important: Those packages may not be up to date with the latest releases. Before submitting a bug report, check the package version and compare that to our latest release then review the changelog to see if the bug has been fixed. Report bugs to the package maintainer rather than directly to Monkeysign if the package is out of date in the distribution.

Also consider that the packages below (apart from the Debian packages) have not been reviewed by the Monkeysign team.

Distribution	Source	Command
Arch Linux	AUR[1]	`pacman -S monkeysign`
Debian	jessie, stretch, sid, ...[2]	`apt-get install monkeysign`
Gentoo	ebuild	`emerge monkeysign`
openSUSE	openSUSE official repository	`zypper in python-monkeysign`
Raspbian	Raspbian pool	`apt-get install monkeysign`
Ubuntu	14.04, 15.04, 15.10, 16.04, ...	`apt-get install monkeysign`

Tip: Please ask package maintainers to build a package for your platform if it is missing above or, if you can package / submit it yourself, please help us with that! If you package Monkeysign, please let us know by filing an issue detailing the distribution name, a link to the package and a command to install it.

Using PIP

You can install Monkeysign with PIP, with the following command:

```
pip install monkeysign
```

Important: Note that 2.1.0 is the first release of Monkeysign published this way. It has not received as much testing as the other methods.

From source

Installing Monkeysign from source is harder, and shouldn't generally be necessary. You may be asked, however, to do that in order to test if your bug is still present in the current release.

Requirements

The following Python packages are required for the GUI to work:

```
python-qrencode python-gtk2 python-zbar python-zbarpygtk
```

If they are not available, the commandline signing tool should still work but doesn't recognize QR codes.

Monkeysign requires a working GnuPG installation.

Downloading

You can fetch Monkeysign with git:

```
git clone https://0xacab.org/monkeysphere/monkeysign.git
```

[1] The AUR package ships with patches that have not been reviewed by the Monkeysign team.
[2] Monkeysign has been in Debian since Debian 6 (*squeeze-backports-sloppy*) and is maintained there as a native package.

Tarballs are also automatically generated on the 0xACAB site for the main branch or you can download tarballs for every past release as well.

You can also find a source tarball from the Debian mirrors here:

```
http://cdn.debian.net/debian/pool/main/m/monkeysign/
```

The `.tar.gz` file has a checksum, cryptographically signed, in the `.dsc` file.

Installing

To install monkeysign from source, run:

```
sudo ./setup.py install --record=install.log
```

Running

It is also be possible to run Monkeysign without installing it, directly from the source tree, with:

```
./scripts/monkeysign
```

and:

```
./scripts/monkeyscan
```

See the *Usage* section for more information on how to use Monkeysign.

Usage

Monkeysign comes in two different interfaces: a commandline interface named *monkeysign* and a graphical interface named *monkeyscan*.

Monkeysign creates a temporary keyring to sign keys, and then encrypts and sends the signature by email to the owner of the key. This makes possible to verify that the holder of the private key (used to decrypt the signature) has also access to the mailbox mentioned in the key.

Note: Make sure you have your email credentials in hand and read the *Sending signed key material* section before starting, as Monkeysign may not know, by default, how to send email.

Tip: If you have problems using Monkeysign, please do report issues and bugs about it, it's a great way of contributing! We also welcome documentation, translation and patches, see *Contribute* for more information.

Monkeysign

The commandline interface should provide you with a complete help file when called with `--help`:

```
monkeysign --help
```

For example, to sign the Monkeysign test key:

```
monkeysign 3F94 240C 918E 6359 0B04  152E 86E4 E70A 96F4 7C6A
```

This will fetch the key from your keyring (or a keyserver) and sign it in a temporary keyring, then encrypt the signature and send it in an email to the owner of the key. Emails can be sent in different ways, documented in *Sending signed key material*.

If the wrong secret key is chosen to sign the key, you can override it with the `--user` option.

Caution: It is important to use Monkeysign with the fingerprint, and *not with the key id*, specially when using it through the Tor network, as a key id can be duplicated easily, unlike fingerprints.

Monkeyscan

The graphical interface (*GUI*) should be self-explanatory, it should be in your regular application menus, or you can call start it form the commandline with:

```
monkeyscan
```

The GUI will show you a *bar code* representing the fingerprint of what Monkeysign thinks is your primary key. You can change that in the *Identity* menu, or by customizing the `default-key` parameter in your `gpg.conf` file.

On the left side, you should see the output of your camera. You can change cameras (if you have more than one) in the *Video device* menu, where you can also turn off the camera altogether.

To exchange fingerprints, you should point the camera at another user's bar code. Monkeysign will detect that user's key fingerprint, fetch the key over the network from keyservers, then ask you for confirmation before signing and sending the email, just like the command line interface. See *Sending signed key material* for more information about how email is sent in Monkeysign in general.

There is a very crude preferences window available in the *Edit* menu. There is work underway to improve it (see 0xA-CAB issue #41), but it should allow you to create a configuration file with your personal settings. See *Configuration files* for more information about this as well.

Sending signed key material

Monkeysign will attempt to send the signed key by email, unless the `--no-mail` argument is specified. In this case, the encrypted key material is shown on the terminal and can then be copy-pasted in the medium of your choice. This is useful, for example, if you use a web email client like Roundcube, Google Mail or similar.

Monkeysign supports many ways of sending emails:

- *Using the system email software (MTA)* (e.g. sendmail or Postfix)
- *Using your normal email client (MUA)* (e.g. Thunderbird or Mutt)
- *Using SMTP* (e.g. connecting directly to your provider)

Also note that the `--tor` option affects how email will be sent, but only when using the SMTP method, as Monkeysign has no way to handle how your *MUA* or *MTA* will talk to the network.

Using the system email software (MTA)

Monkeysign, by default, assumes you have a local *MTA* installed as `sendmail`. If it is not, you can specify the path to a `sendmail` compatible program with the `--mta` option. Such a program should accept the complete message on standard input, and the recipient is passed on the commandline in place of the `%(to)s` argument.

Note that it is uncommon for workstations and laptops to have a working *MTA* installed: this is more commonly done on servers, and unless you know what you are doing, you are more likely to want to talk to your existing email client, your *MUA*.

Using your normal email client (MUA)

Therefore, to properly send email on your workstation, you may need to tell Monkeysign how to use your regular email client, your *MUA*. For this, you can use the `--mua` option.

By default, Monkeysign will try to figure out your default email client when you use the `--mua` flag is used without argument. This will in turn call the `xdg-email` command which automatically uses your configured email client correctly:

```
monkeysign --mua [...]
```

Your default mail client can be modified in your desktop environment control panel, or with the `xdg-mime` command, for example this will set Thunderbird as your default email client:

```
xdg-mime default thunderbird.desktop x-scheme-handler/mailto
```

You can also specify your own email client on the fly. Here are few examples of known working configurations.

- Thunderbird:

```
monkeysign --mua "thunderbird -compose to=%(to)s,subject=%(subject)s,body=
↪%(body)s,attachment=%(attach)s" [...]
```

> **Note:** Thunerbird fails to respect the attachment parameter in versions before 52.1.1, see Debian BTS #837771 for more details.

- Mutt:

```
monkeysign --mua "mutt -a %(attach)s -s %(subject)s -i %(body)s %(to)s" [...]
```

Finally, note that you need to confirm when you are finished writing the actual email. This is because we cannot tell when the email is sent, because a lot of software (especially Thunderbird) return before the email is sent, see Debian BTS #677430 for more information about this issue.

Note: Essentially, the difference between `--mta` and `--mua` is that the complete message is piped through the *MTA* command whereas it is passed as an argument on the commandline for *MUA* commands. Also, the `--mta` command expands only the `%(to)s` parameter, whereas the `--mua` command expands `%(attach)s`, `%(subject)s`, `%(body)s` and `%(to)s`.

Note that when a `--mua` is used, only the key material is encrypted: the body of the email is sent in the clear. This is because Monkeysign cannot control how the attachment layout in the *MUA* in a standard way.

Furthermore, it may be more difficult for the end-user to import the key when it was sent with a *MUA*, as the recipient's own *MUA* may not know how to both decrypt *and* import the key at the same time. The *MTA* method doesn't have this problem because of MIME encapsulation. See also 0xACAB issue #7 for a broader technical discussion about the `--mua` implementation.

Using SMTP

Note that you can also send email using your provider's SMTP server directly, turning Monkeysign into a *MUA* itself. For example:

```
monkeysign --smtp=mail.example.com:587 --smtpuser=john [fingerprint of OpenPGP key to␣
↪sign]
```

In the above, Monkeysign will attempt to connect to the `mail.example.com` SMTP server over the submission port (`587`), attempt to upgrade the connection securely (using `STARTTLS`) and use the *john* username. Password will be prompted securely.

Tip: To use a raw `TLS` connection, you can also use the `--tls` flag.

Tip: You can also try to deliver email over Tor network with the `--tor` option. Be be aware that a lot of email providers block Tor exit nodes for spam control. You may need to use your provider's hidden service to workaround those issues. Ask your email provider for Tor support if you have problems with the SMTP method.

Configuration files

Monkeysign will read `/etc/monkeysign.conf` and `~/.config/monkeysign.conf` (in that order) for configuration options. Each option can be specified on its own line. Lines starting with the pound sign (`#`) are ignored as comments. A configuration file can be generated with the `--save` option, or through the preferences window in the GUI. Here is a sample configuration file:

```
# use my SMTP server to send email
smtpserver=smtp.example.com:587
# this is my username, password is securely prompted interactively
smtpuser=john
# be more verbose
verbose
```

As you can see, flags like `--verbose` are simply specified on their own, while options with arguments need to be seperated with an equal (=) sign.

Support

If you have problems or question using Monkeysign, there are several options at your disposal:

- Try to troubleshoot the issue yourself
- Write to the mailing list
- Chat on IRC
- File bug reports

We of course welcome other contributions like documentation, translations and patches, see the *Contribute* guide for more information on how to contribute to the project.

Troubleshooting

The basic way to troubleshoot Monkeysign is to run the same command as you did when you had an error with the `--verbose` or, if that doesn't yield satisfactory results, with the `--debug` output.

Note: The debug output outputs a lot of information, as it shows the OpenPGP key material as it is exchanged with GnuPG. It may be confusing for new users.

If you suspect there is a bug in Monkeysign specific to your environment, you can also try to see if it is reproducible within the test suite with `monkeysign --test`. From there, you can either file a bug report or try to fix the issue yourself, see the *Contribute* section for more information.

Otherwise, see below for more options to get support.

Mailing list

Anyone can write to the mailing list at monkeysphere@lists.riseup.net. You can browse the archives before posting to see if your question has already been answered. Thanks to Riseup.net for graciously hosting our mailing list.

Tip: We encourage you to donate to Riseup to support the Monkeysign project, as we use several parts of their infrastructure to develop Monkeysign.

Note that the mailing list is for the larger Monkeysphere project so if you subscribe, you should expect discussions to go beyond only Monkeysign. Furthermore, when you write to the mailing list, you should explicitly mention that you are talking about Monkeysign.

Chat

We are often present in realtime in the `#monkeysphere` channel of the OFTC network. You can join the channel using this link or this web interface.

Bug reports

We want you to report bugs you find in Monkeysign. It's an important part of contributing to a project, and all bug reports will be read and replied to politely and professionally.

We are using 0xACAB.org's Gitlab instance to manage issues, and this is where bug reports should be sent. Some issues are also documented by Debian users directly.

Tip: A few tips on how to make good bug reports:

- Before you report a new bug, review the existing issues in the online issue tracker and the Debian BTS for Monkeysign to make sure the bug has not already been reported elsewhere.
- The first aim of a bug report is to tell the developers exactly how to reproduce the failure, so try to reproduce the issue yourself and describe how you did that.
- If that is not possible, just try to describe what went wrong in detail. Write down the error messages, especially if they have numbers.
- Take the necessary time to write clearly and precisely. Say what you mean, and make sure it cannot be misinterpreted.

- Include the output of `monkeysign --test`, `monkeysign --version` and `monkeysign --debug` in your bug reports. See the issue template for more details about what to include in bug reports.

If you wish to read more about issues regarding communication in bug reports, you can read How to Report Bugs Effectively which takes about 30 minutes.

Warning: The output of the `--debug` shows public key material used by Monkeysign. Special efforts have been made so that private key material is never displayed (or in fact accessed directly or copied) but you may want to avoid publicly disclosing which keys you are signing because that can reveal your social graph. If you are confident the signed user will publish the results on the public keyservers, this is not much of a concern. But otherwise, you should leave that decision to that user. This is particularly relevant if you do *not* want to publicly certify this (e.g. if you are using the `--local` flag). Do review the output before sending it in bug reports.

Debian BTS

You can also report bugs by email over the Debian BTS, even if you are not using Debian. Use the `reportbug` package to report a bug if you run Debian (or Ubuntu), otherwise send an email to `submit@bugs.debian.org`, with content like this:

```
To: submit@bugs.debian.org
From: you@example.com
Subject: fails to frobnicate

Package: monkeysign
Version: 1.0

Monkeysign fails to frobnicate.

I tried to do...

I was expecting...

And instead I had this backtrace...

I am running Arch Linux 2013.07.01, Python 2.7.5-1 under a amd64
architecture.
```

See also the complete instructions for more information on how to use the Debian bugtracker. You can also browse the existing bug reports in the Debian BTS for Monkeysign there.

Contribute

This section explains the various ways users can participate in the development of Monkeysign, or get support when they find problems.

Code of conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting one of the persons *listed below* individually. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. Project maintainers are obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Project maintainers are encouraged to follow the spirit of the Django Code of Conduct Enforcement Manual when receiving reports.

Contacts

The following people have volunteered to be available to respond to Code of Conduct reports. They have reviewed existing literature and agree to follow the aforementioned process in good faith. They also accept OpenPGP-encrypted email:

- Antoine Beaupré <anarcat@debian.org>
- Daniel Kahn Gillmor <dkg@fifthhorseman.net>

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4.

Changes

The Code of Conduct was modified to refer to *project maintainers* instead of *project team* and small paragraph was added to refer to the Django enforcement manual.

Note: We have so far determined that writing an explicit enforcement policy is not necessary, considering the available literature already available online and the relatively small size of the Monkeysign community. This may change in the future if the community grows larger.

This code of conduct was adopted in 2016 by the Monkeysign maintainers, see 0xACAB issue #54 for more details about the discussion.

Support schedule

First, to know a bit more about the version you are using, understand that we adhere to Semantic Versioning, which is:

> Given a version number MAJOR.MINOR.PATCH, increment the:
>
> - MAJOR version when you make incompatible API changes,
> - MINOR version when you add functionality in a backwards-compatible manner, and
> - PATCH version when you make backwards-compatible bug fixes.
>
> Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

The 2.0.x branch is featured in Debian Jessie and Ubuntu Xenial and is therefore be maintained for security fixes for the lifetime of those releases or of any other distribution that picks it up.

Most development and major bug fixes are done directly in the 2.x branch and published as part of minor releases, which in turn become supported branches.

Major, API-changing development will happen on the 3.x branch.

Those milestones are collaboratively tracked on 0xACAB.

Branches status

Each branch may be in one of those states:

- Development: the development branch is where most of the new features are implemented. Consequently, new features and changes may inadvertently break things.
- Supported: The branch is supported, but no further development will be made on the branch. Only critical issues and security fixes are performed.
- Deprecated: users are strongly encouraged to upgrade to later versions. No further updates perform except for critical security issues.
- Abandoned: the branch is completely abandoned. No further updates will ever be performed on the branch, security or otherwise.

Branch	Status	Notes
0.x	Abandoned	explicitly unsupported.
1.x	Deprecated	supported until the release of Debian jessie, now only in Debian LTS
2.0.x	Supported	supported until the end of Debian jessie
2.1.x	Abandoned	short lived support branch, superseded by 2.2.x
2.2.x	Supported	supported until the end of Debian stretch
2.x	Development	new releases performed here, schedule to be clarified

We try to keep the number of "supported" and "deprecated" branches to two each, which means it is likely a "deprecated" branch gets abandoned when a "supported" branch gets "deprecated".

If you are interested in supporting one of those branches beyond the current state, we would be glad to welcome you on the team. *Contact us*!

See also *History* for more information on past releases.

Documentation

We love documentation!

We maintain the documentation in the Git repository, in RST format. Documentation can be edited directly on the website and built locally with Sphinx with:

```
cd doc ; make html
```

The Sphinx project has a good tutorial on RST online. Documentation is automatically generated on RTD.io.

Translation

Monkeysign is translated using the standard Gettext translation system. Translation files are located in the source tree, in the `po/` subdirectory and can be edited using standard translation tools or even a regular text editor. A new translation for your locale can be created with the `msginit` command, see the gettext manual for more information about how to use gettext directly.

You can also use the Weblate web interface to translate Monkeysign directly in your browser, without having to install any special software. New translations from Weblate need to be updated in our source tree by hand, so do let us know if you make a new translation, by filing an issue in our online issue tracker.

Note: We have chosen Weblate instead of other solutions because it integrates well with our git-based workflow: translations on the site are stored as commits in the git repository, and the server is just another remote that we can

merge directly. It also merges our changes automatically and so requires minimal work on our part. We have also considered tools like Transifex (proprietary) and Pootle (no public instance, requires us to run our own).

Tip: We encourage our users and developers to support Weblate's development. Thank you to Weblate's people for hosting our project for free!.

Bug reports

We want you to report bugs you find in Monkeysign. It's an important part of contributing to a project, and all bug reports will be read and replied to politely and professionally. See the *Support* section for more information about troubleshooting and bug reporting.

Bug triage

Bug triage is a very useful contribution as well. You can review the issues on 0xACAB or in the Debian BTS for Monkeysign. What needs to be done is, for every issue:

- try to reproduce the bug, if it is not reproducible, tag it with `unreproducible`
- if information is missing, tag it with `moreinfo`
- if a patch is provided, tag it with `patch` and test it
- if the patch is incomplete, tag it with `help` (this is often the case when unit tests are missing)
- if the patch is not working, remove the `patch` tag
- if the patch gets merged into the git repository, tag it with `pending`
- if the feature request is not within the scope of the project or should be refused for other reasons, use the `wontfix` tag and close the bug (with the `close` command or by CC'ing `NNNN-done@bugs.debian.org`)
- feature requests should have a `wishlist` severity

Those directives apply mostly to the Debian BTS, but some tags are also useful in the 0xACAB site. See also the more complete directives on how to use the Debian BTS.

Patches

Patches can be submitted through merge requests on the Gitlab site. You will need to contact the 0xACAB staff to request access before you can create a fork and a merge request.

If you prefer old school, offline email systems, you can also use the Debian BTS, as described above, or send patches to the mailing list for discussion.

Some guidelines for patches:

- A patch should be a minimal and accurate answer to exactly one identified and agreed problem.
- A patch must compile cleanly and pass project self-tests on at least the principle target platform.
- A patch commit message must consist of a single short (less than 50 characters) line stating the a summary of the change, followed by a blank line and then a description of the problem being solved and its solution, or a reason for the change. Write more information, not less, in the commit log.

Maintainers should not merge their own patches unless there is no response from other maintainers within a reasonable time frame (1-2 days).

Note: Those guidelines were inspired by the Collective Code Construct Contract. The document was found to be a little too complex and hard to read and wasn't adopted in its entirety. See those discussions for more information.

Unit tests

Unit tests should be ran before sending patches. They can be ran with `monkeysign --test` (starting from Monkeysign 2.1.4, previously it was `./test.py` and only from the source tree).

The tests expect a unicode locale, so if you do not have that configured already, set one like this, otherwise a part of the test suite will be skipped:

```
export LANG=C.UTF-8
monkeysign --test
```

It is possible that some keys used in the tests expire. The built-in keys do not have specific expiry dates, but some keys are provided to test some corner cases and *those* keys may have new expiration dates. Those tests should be skipped when the key expire, but the keys should eventually be renewed.

To renew the keys, try:

```
mkdir ~/.gpg-tmp
chmod 700 ~/.gpg-tmp
gpg --homedir ~/.gpg-tmp --import 7B75921E.asc
gpg --homedir ~/.gpg-tmp --refresh-keys 8DC901CE64146C048AD50FBB792152527B75921E
gpg --homedir ~/.gpg-tmp --export-options export-minimal --armor --export␣
↪8DC901CE64146C048AD50FBB792152527B75921E > 7B75921E.asc
```

It is also possible the key is just expired and there is no replacement. In this case the solution is to try and find a similar test case and replace the key, or simply skip that test.

Debian packaging

The Debian package requires backports of `dh-python` to operate properly, otherwise you will get errors like Debian BTS #839687:

```
LookupError: setuptools-scm was unable to detect version for '/tmp/buildd-...'.
```

A workaround is to hardcode the version with:

```
SETUPTOOLS_SCM_PRETEND_VERSION=x.y.z
```

Release process

To build a Monkeysign release, you will need to have a few tools already installed, namely the Python packages `wheel`, `setuptools` and `setuptools-scm`. We also assume you use the following Debian packages, although you may be able to work around those: `devscripts`, `git`, `git-buildpackage`, `pip` and `twine`. In Debian, this should get you started:

```
sudo apt install python-wheel python-setuptools python-setuptools-scm devscripts git␣
↪git-buildpackage python-pip twine
```

1. make sure tests pass:

   ```
   ./scripts/monkeysign --test
   ```

2. create release notes with:

   ```
   git-dch
   dch -D unstable
   ```

3. commit the results:

   ```
   git commit -m"prepare new release" -a
   ```

4. create a signed and annotated tag:

   ```
   git tag -s x.y.z
   ```

5. build and test Debian package:

   ```
   git-buildpackage
   dpkg -i ../monkeysign_*.deb
   monkeysign --version
   monkeysign --test
   monkeyscan
   dpkg --remove monkeysign
   ```

6. build and test Python "wheel":

   ```
   python setup.py bdist_wheel
   pip install dist/*.whl
   monkeysign --version
   monkeysign --test
   monkeyscan
   pip uninstall monkeysign
   ```

7. push commits and tags to the git repository:

   ```
   git push
   git push --tags
   ```

8. publish Python "wheel" on PyPI:

   ```
   twine upload dist/*
   ```

9. upload Debian package:

   ```
   dput ../monkeysign*.changes
   ```

10. add announcement on website, IRC channel and mailing list: monkeysphere@lists.riseup.net
11. on 0xACAB: close the current milestone, create the next one and edit the release notes on the tag

History

A first prototype of Monkeysign was created during the 2010 The Next HOPE in New York City by Jérôme Charaoui, after a discussion with Daniel Kahn Gillmor and Antoine Beaupré.

During the following HOPE conference in 2012, the happy group met again and this time Antoine pushed the project much further. On the train ride back home, he implementing a complete GnuPG compatibility layer that imported keys, signed UIDs and so on, unaware of the existence of the *python-gnupg* project. This led to the publication of the first 0.1 release, which was already a good `caff` replacement, with a GUI and qr-code support.

After one more year of development and testing, Antoine released the first stable 1.0 release in 2013 with SMTP support, unitests and most features we now take for granted in Monkeysign. The 1.x branch was fairly short-lived, a 2.0.0 release was published in 2014 with simplified interface, image files support and more improvements. This release was the first long term support branch, issued to coincide with the Debian Jessie release.

Since then, a 2.1.x series was published to support GnuPG 2.1, and 2.2.x was released with support for Tor.

See *Support schedule* for more information about supported branches.

Detailed changelog

Here is the complete historical record of all known Monkeysign releases at the time of writing. More details about changes is also available in the git repository.

```
monkeysign (2.2.3) unstable; urgency=medium

  [ Simon Fondrie-Teitler ]
  * Don't escape percent signs that are actually required in default mua command

  [ Antoine Beaupré ]
  * some small improvements to the bug issue template
  * create 2.2.x branch officially
  * silence errors in test suite with GnuPG 2

 -- Antoine Beaupré <anarcat@debian.org>  Tue, 24 Jan 2017 15:40:35 -0500

monkeysign (2.2.2) unstable; urgency=medium

  [ Antoine Beaupré ]
  * explicitly depend on socks, seems like pybuild doesn't puck up the
    depends (Closes: #847716)
  * forgot some future tests failures (Closes: #841115)
  * properly redirect version information
  * mention --test in bug report guidelines
  * clarify support schedule, fix typos
  * abandon 2.1.x, tell people how to support more
  * indicate that you need to request access to create merge requests
  * document the new test skipping features
  * give proper credits to documenters
  * add credits section
  * fix trove classifier
  * output the parsed qrcode data when verbose
  * do not load default config files in tests
  * adopt covenant code of conduct
  * patches merging guidelines
  * refer to modernPGP manuals
  * move code of conducts contacts to a special section
```

```
  [ Simon Fondrie-Teitler ]
  * Add right click menu with print/save to qr code
  * Don't attempt to sign a user's own key
  * Make message more friendly
  * add test for signing one's own key
  * lowercase k in OpenPGPkey __repr__
  * Add Simon to authors file

  [ Tobias Mueller ]
  * gpg: Use os.path.expanduser instead of the environment variable

 -- Antoine Beaupré <anarcat@debian.org>  Thu, 15 Dec 2016 11:04:13 -0500

monkeysign (2.2.1) unstable; urgency=medium

  * fix socks dependency specification: it is a runtime, not just
    build-time, dependency
  * mark as production-ready in python classification
  * skip another test that requires network during build
  * run CI tests with --debug to ease future debugging

 -- Antoine Beaupré <anarcat@debian.org>  Sat, 15 Oct 2016 09:18:21 -0400

monkeysign (2.2.0) unstable; urgency=medium

  * fix tests with Debian CI
  * fix FTBS errors in reproducible builds due to test suite failing in
    the future
  * do not STARTTLS on already secure (TLS) connexions
  * enable tor support with --tor flag
  * handle SMTP conversations better
  * add history section to documentation to publish this changelog more
    widely
  * document branches status and deprecate 2.1.x branch
  * improve email usage documentation

 -- Antoine Beaupré <anarcat@debian.org>  Tue, 11 Oct 2016 11:29:10 -0400

monkeysign (2.1.4) unstable; urgency=medium

  * --local now implies --no-mail (Closes: #719242)
  * ship tests with program, accessible with --test parameter
  * stop hardcoding version numbers in code, use setuptools-scm instead
  * enable tests at build time and Debian CI (autopkgtest)
  * complete GnuPG 2.1 support: test suite now passes!

 -- Antoine Beaupré <anarcat@debian.org>  Mon, 03 Oct 2016 16:18:07 -0400

monkeysign (2.1.3) unstable; urgency=medium

  * add explicit build-dep on gnupg (Closes: #839355)

 -- Antoine Beaupré <anarcat@debian.org>  Sun, 02 Oct 2016 17:17:03 -0400

monkeysign (2.1.2) unstable; urgency=medium

  * reroll release: forgot to bump version number in ode
```

```
  * upload to pypi before debian, which will notice those errors in the future

 -- Antoine Beaupré <anarcat@debian.org>  Wed, 28 Sep 2016 09:17:20 -0400

monkeysign (2.1.1) unstable; urgency=medium

  * properly transition monkeysign-doc packages to ensure upgrades work
    (Closes: #839043)
  * add monkeysign-doc to Suggests
  * remove obsolete BUILD_TIMESTAMP, especially now that the manpage
    generation was rewritten without timestamps
  * improve release process and install documentation, remove presentation
  * forgot to close a bunch of issues in 2.1.0 release:
    * Monkeyscan fails at launch (Closes: #773970)
    * expiry date in epoch time is not human readable (Closes: #760139)
    * make builds reproducible (Closes: #784602)

 -- Antoine Beaupré <anarcat@debian.org>  Wed, 28 Sep 2016 08:18:24 -0400

monkeysign (2.1.0) unstable; urgency=medium

  * new minor release for new features and lots of bugfixes, outline:
    * GnuPG 2.1 support
    * better handling of corner cases (revoked or expired key material,
      large webcams) and better error messages)
    * better SMTP support (no cleartext, SSMTP)
    * move everything to 0xACAB.org to ease collaboration
    * expand and convert documentation to reStructured Text and ship it in
      a -doc package
    * command to sendmail customizable through --mta (message piped
      through stdin) or --mua (encrypted key attached on the commandline)
    * space-separated fingerprints allowed for -u, which means -u needs to
      be separated from the signed fingerprint with -- now
    * configuration file support, which is written with --save
    * crude preferences window in GUI
    * detailed changelog below - this is the result of 2 years of work!

  [ Antoine Beaupré ]
  * import my personal key renewal to unbreak tests
  * import zack's key renewal
  * forbid sending passphrase in cleartext
  * better explain that STARTTLS is used
  * SSMTP support
  * port to argparse, which somewhat broke the manpages
  * allow space-separated fingerprints for -u (Closes: #720050)
  * MUA support
  * make sendmail command customizable through --mta
  * make copy-paste message encrypted (Closes: #833605)
  * handle improperly encoded UIDs (Closes: #736629)
  * copy public keys for all secret keys found (Closes: #721599)
  * skip keys without uids (Closes: #723152)
  * set a size for the webcam to avoid too large videos (Closes: #723154)
  * add more tests for signing revoked uids
  * add unit test for expired subkeys
  * accommodate gitlab's naming conventions
  * move to 0xacab.org for issues, removing bugs-everywhere
  * convert markdown documents to RST
  * merge the website in the main documentation
```

```
  * expand documentation: support schedule, semantic versioning, PyPI, etc
  * update urls for openkeychain, mark as compatible
  * reshuffle test suite so we make sure it tests the local code
  * style fixes
  * fix a transient error in unit tests
  * mention tests need a unicode locale
  * fix monkeysign detection in source dir
  * detect revoked keys and do not use them to sign keys
    (Closes: #766129, #773896)
  * fix lintian warning by specifying copyright version
  * don't try to remove non-existent video device, and clarify error message
  * output --version to stdout and don't make it an error
  * properly raise exceptions when copying gpg.conf fails
  * make sure ui calling sequence is correct in sign_key
  * use ttyname instead of the tty command
  * fix potential vulnerability in msgfmt parser
  * review code for security issues with bandit
  * handle missing MTA better, see 0xACAB #39
  * use full path to sendmail, see 0xACAB #39
  * clarify that without smtp, we use the default --mta
  * fix whitespace issues in revoked patches
  * add new trust state, `empty`
  * properly fetch secret key material everywhere
  * seek out secret keys first
  * properly show output of runtime errors
  * include standard debugging information on backtrace
  * add hook to show detailed version information in reportbug (see 0xACAB #39)
  * always enable --verbose when --debug is enabled
  * configuration file support, which is written with --save
  * crude preferences window in GUI

  [ Kristian Fiskerstrand ]
  * ui.py: Make sure to use smtplib namespace

  [ Tobias Mueller ]
  * Calculated whether a key has expired based on the parsed expiry
  * gpg: Added a __repr__ for UIDs
  * gpg: Added a __repr__ for OpenPGPKeys
  * Added GnuPG 2.1 compatibility reg. its colon output
  * gpg: Fixed up the key parsing for secret keys
  * gpg: Make a full datetime, instead of epoch, for expiry
  * msgfmt: Increase Python3 compatibility by removing "L" suffix
  * translation: Use print() for increased python3 compatibility
  * gpg: Implemented revoked for OpenPGP Keys
  * gpg: Implemented revoked for OpenPGP UIDs

  [ Daniel Kahn Gillmor ]
  * use new-style gbp.conf
  * make monkeysign build reproducibly

  [ Michael R. Lawrence ]
  * Translated using Weblate (Italian)
  * Translated using Weblate (French)

  [ Michal Čihař ]
  * Translated using Weblate (Czech)

  [ Ahmed El Azzabi ]
```

```
  * Translated using Weblate (French)

  [ Gonzalo Exequiel Pedone ]
  * Translated using Weblate (Spanish)

  [ Jerome Charaoui ]
  * Remove bugseverwhere data and migrate issues to 0xacab.org
  * Ignore irrelevant gpg errors (Closes: #736548)

  [ Ramakrishnan Muthukrishnan ]
  * Improve the error message when signing an already signed key.
  * improve unit tests for already signed keys and keep previous check

  [ emma peel ]
  * various improvements to the documentation

 -- Antoine Beaupré <anarcat@debian.org>  Tue, 13 Sep 2016 13:37:50 -0400

monkeysign (2.0.2) unstable; urgency=medium

  * this patch releases fixes critical issues...
  * reported in the Debian BTS:
    * encode prompt properly before prompting (closes: #771032)
    * try to handle error when import actually works in GTK UI
      (closes: #770900)
    * improve debugging by wrapping all writes to gnupg in debug
    * use the proper index when selecting key to sign
      (closes: #771034)
  * reported on the Monkeysphere mailing list:
    * hotfix: properly verify the image signature file
    * hotfix: disable scrolling in qrcode window
    * don't try to remove non-existent video device, and clarify error
      message
    * output --version to stdout and don't make it an error
  * those fix FTBS issues:
    * fix tests after cd4e18c: guess encoding properly
    * update zack's key so tests succeed again

  * and this makes sure this package will be easier to support for the
    lifetime of jessie
    * improve error handling again: distinguish different failure cases
      and clearly transmit GPG errors

 -- Antoine Beaupré <anarcat@debian.org>  Mon, 01 Dec 2014 21:03:56 -0500

monkeysign (2.0.1) unstable; urgency=medium

  * hot patch release while we still can before jessie:
    * fix tests under GnuPG 2.x
    * improve usage to clarify -u, --cert-level and --to
    * fix version number to include patch release

 -- Antoine Beaupré <anarcat@debian.org>  Mon, 20 Oct 2014 22:24:37 -0400

monkeysign (2.0.0) unstable; urgency=medium

  * new features:
    * implement qrcode image import, to allow people without webcams to
```

```
      import pictures from a trusted camera - the images must be signed with
      a detached signature on pain of a ugly warning with instructions.
    * move to bugs-everywhere instead of that crazy TODO file
    * udate french translation
  * usability improvements:
    * interface simplified: only the qrcode and webcam with instructions
    * all options moved to menus, including the print/save buttons, the
      video and identity dropdowns
    * properly handle exceptions in gtk UI
    * avoid duplicate camera listing and display nicer name (Closes: #718796)
    * create a set of mockups for a UI redesign and API documentation
      rendered at http://monkeysign.readthedocs.org/
  * bug fixes:
    * fix "Content-description" to be more useful (Closes: #723677)
    * support monkeysign --version", thanks to Gabriel Fillion (Closes: #725113)
    * add debugging info from smtp connection, thanks to Gabriel Filion
      (Closes: #756540)
    * some improvements were done in the GnuPG library to work around
      certain GnuPG corner cases and describe problems better
    * install monkeyscan command as a symlink properly (Closes: #743150)
  * switch to long term support strategy for the 2.0.x release in
    preparation for Debian Jessie

 -- Antoine Beaupré <anarcat@debian.org>  Sat, 18 Oct 2014 13:25:54 -0400

monkeysign (1.2) unstable; urgency=medium

  * improve python 3 compatibility, partially (Closes: #725059)
  * update translation strings
  * spanish translation, thanks to lilbit
  * partial french translation
  * Czech translation, thanks to Michal Čihař
  * Bug fix: "build_slides fails of two reasons", thanks to Felix Dreissig
    (Closes: #738731).
  * Bug fix: "build_manpage only works because of PyGTK encoding changes",
    thanks to Felix Dreissig (Closes: #738730).
  * Bug fix: "build_trans fails if called seperately", thanks to Felix
    Dreissig (Closes: #738732).

 -- Antoine Beaupré <anarcat@debian.org>  Thu, 28 Aug 2014 20:23:57 -0700

monkeysign (1.1) unstable; urgency=low

  [Antoine Beaupré]
  * improved SMTP support:
    * SMTP username and passwords can be passed as commandline arguments
    * SMTP password is prompted if not specified
    * use STARTTLS if available
    * enable SMTP debugging only debugging is enabled
  * show the unencrypted email with --no-mail (Closes: #720049)
  * warn when gpg-agent is not running or failing (Closes: #723052)
  * set GPG_TTY if it is missing (Closes: #719908)
  * bail out on already signed keys (Closes: #720055)
  * mention monkeyscan in the package description so it can be found more
    easily
  * fix python-pkg-resources dependency
  * don't show backtrace on control-c
  * add missing files to .gitignore (Closes: #724007)
```

```
  * ship with a neat little slideshow to make presentations

  [Philip Jägenstedt]
  * fix some typos (Closes: #722964)
  * add --cert-level option (Closes: #722740)

 -- Antoine Beaupré <anarcat@debian.org>  Tue, 01 Oct 2013 00:22:30 +0200

monkeysign (1.0) unstable; urgency=low

  * stop copying secrets to the temporary keyring
  * make sure we use the right signing key when specified
  * signatures on multiple UIDs now get properly sent separately
    (Closes: #719241)
  * this includes "deluid" support on the gpg library
  * significantly refactor email creation
  * improve unit tests on commandline scripts, invalid (revoked) keys and
    timeout handling
  * provide manpages (Closes: #716674)
  * avoid showing binary garbage on export when debugging
  * properly fail if password confirmation fails
  * user interfaces now translatable
  * accept space-separated key fingerprints
  * fix single UID key signing
  * proper formatting of UIDs with comments (removed) and spaces (wrapped)
    for emails

 -- Antoine Beaupré <anarcat@debian.org>  Wed, 14 Aug 2013 20:51:44 -0400

monkeysign (0.9) unstable; urgency=low

  * refactor unit tests again to optimise UI tests and test mail generation
  * fix error handling in encryption/decryption (Closes: #717622)
  * rename msign-cli to monkeysign and msign to monkeyscan (Closes: #717623)
  * handle interruptions cleanly when choosing user IDs (see: #716675)

 -- Antoine Beaupré <anarcat@debian.org>  Tue, 23 Jul 2013 10:56:50 -0400

monkeysign (0.8) unstable; urgency=low

  * refactor unit test suite to allow testing the commandline tool
    interactively
  * don't fail on empty input when choosing uid (Closes: #716675)
  * we also explain how to refuse signing a key better
  * optimise network tests so they timeout (so fail) faster

 -- Antoine Beaupré <anarcat@debian.org>  Wed, 17 Jul 2013 22:52:02 -0400

monkeysign (0.7.1) unstable; urgency=low

  * fix binary package dependency on python
  * update to debhelper 9
  * update to standards 3.9.4, no change

 -- Antoine Beaupré <anarcat@debian.org>  Sun, 07 Jul 2013 09:58:56 -0400

monkeysign (0.7) unstable; urgency=low
```

```
  * fix crash when key not found on keyservers
  * use a proper message in outgoing emails
  * unit tests extended to cover user interface
  * import keys from the local keyring before looking at the keyserver
  * fix print/save exports (thanks Simon!)
  * don't depend on a graphical interface
  * update copyright dates and notices
  * mark as priority: optional instead of extra

 -- Antoine Beaupré <anarcat@debian.org>  Sat, 06 Jul 2013 01:07:28 -0400

monkeysign (0.6) unstable; urgency=low

  * fix warnings in the graphical interface
  * make qr-code detection be case-insensitive
  * fix syntax error
  * follow executable renames properly

 -- Antoine Beaupré <anarcat@debian.org>  Sat, 06 Oct 2012 16:08:48 +0200

monkeysign (0.5) unstable; urgency=low

  * non-exportable signatures (--local) support
  * simplify the monkeysign-scan UI
  * rename monkeysign-scan to msign and monkeysign-cli to msign-cli to
    avoid tab-completion conflict with monkeysphere executables, at the
    request of Monkeysphere developers
  * usability: make sure arguments are case-insensitive
  * fix email format so it's actually readable

 -- Antoine Beaupré <anarcat@debian.org>  Fri, 05 Oct 2012 11:14:37 +0200

monkeysign (0.4) unstable; urgency=low

  * merge display and scanning of qrcodes
  * really remove remaining pyme dependency
  * list key indexes to allow choosing more clearly
  * copy the gpg.conf in temporary keyring
  * fix keyserver operation in GUI
  * implement UID choosing in GUI

 -- Antoine Beaupré <anarcat@debian.org>  Wed, 01 Aug 2012 02:33:29 -0400

monkeysign (0.3) unstable; urgency=low

  * allow keyserver to be enabled while not specified
  * do not set an empty keyserver, fixing weird keyserver errors on -scan
  * fix window reference in UI, spotted by dkg
  * mark this as architecture-independent, spotted by dkg
  * make setup executable
  * reference new homepage
  * API change: functions return false instead of raising exceptions
  * fix multiple keys listing support

 -- Antoine Beaupré <anarcat@debian.org>  Thu, 26 Jul 2012 12:41:54 -0400

monkeysign (0.2) unstable; urgency=low
```

```
  * only load information from private keys when doing key detection
  * add debugging in key choosing algorithm
  * import private keyring even in dry-run
  * properly import re, fixing a crash
  * add usage for monkeysign-scan
  * fixup modules list so that the package actually works
  * make this not crash completely if there's no video
  * improve short description so that it matches 'key signing'
  * fix dependencies
  * fix typo, noticed by micah

 -- Antoine Beaupré <anarcat@debian.org>  Sun, 22 Jul 2012 13:38:00 -0400

monkeysign (0.1) unstable; urgency=low

  * Initial Release.

 -- Antoine Beaupré <anarcat@debian.org>  Sat, 21 Jul 2012 12:05:59 -0400
```

API documentation

GnuPG API

Native Python / GPG API

This API was written to replace the GPGME bindings because the GPGME API has a few problems:

1. it is arcane and difficult to grasp
2. it is very closely bound to the internal GPG data and commandline structures, which are quite confusing
3. GPGME doesn't actually talk to a GPG library, but interacts with GPG through the commandline
4. GPGME developers are not willing to extend GPGME to cover private key material management and consider this is outside the scope of the project.

The latter two points are especially problematic for this project, and I have therefore started working on a replacement.

Operations are performed mostly through the Keyring or KeyringTmp class (if you do not want to access your regular keyring but an empty temporary one).

This is how you can access keys, which are represented by the OpenPGPkey datastructure, but which will not look in your keyring or on the keyservers itself without the Keyring class.

It seems that I have missed a similar project that's been around for quite a while (2008-2012):

https://code.google.com/p/python-gnupg/

The above project has a lot of similarities with this implementation, but is better because:

1. it actually parses most status outputs from GPG, in a clean way
2. uses threads so it doesn't block
3. supports streams
4. supports verification, key generation and deletion
5. has a cleaner and more complete test suite

However, the implementation here has:

1. key signing support
2. a cleaner API

Error handling is somewhat inconsistent here. Some functions rely on exceptions, other on boolean return values. We prefer exceptions as it allows us to propagate error messages to the UI, but make sure to generate a RuntimeError, and not a ProtocolError, which are unreadable to the user.

class monkeysign.gpg.**Context**
Python wrapper for GnuPG

This wrapper allows for a simpler interface than GPGME or PyME to GPG, and bypasses completely GPGME to interoperate directly with GPG as a process.

It uses the gpg-agent to prompt for passphrases and communicates with GPG over the stdin for commnads (–command-fd) and stdout for status (–status-fd).

build_command(*command*)
internal helper to build a proper gpg commandline

this will add relevant arguments around the gpg binary.

like the options arguments, the command is expected to be a regular gpg command with the – stripped. the – are added before being called. this is to make the code more readable, and eventually support other backends that actually make more sense.

this uses build_command to create a commandline out of the 'options' dictionary, and appends the provided command at the end. this is because order of certain options matter in gpg, where some options (like –recv-keys) are expected to be at the end.

it is here that the options dictionary is converted into a list. the command argument is expected to be a list of arguments that can be converted to strings. if it is not a list, it is cast into a list.

call_command(*command*, *stdin=None*)
internal wrapper to call a GPG commandline

this will call the command generated by build_command() and setup a regular pipe to the subcommand.

this assumes that we have the status-fd on stdout and command-fd on stdin, but could really be used in any other way.

we pass the stdin argument in the standard input of gpg and we keep the output in the stdout and stderr array. the exit code is in the returncode variable.

we can optionnally watch for a confirmation pattern on the statusfd.

expect(*fd*, *pattern*)
look for a specific GNUPG status on the next line of output

this is a stub for expect()

expect_pattern(*fd*, *pattern*)
make sure the next line matches the provided pattern

in contrast with seek_pattern(), this will *not* skip non-matching lines and instead raise an exception if such a line is found.

this therefore looks only at the next line, but may also hang like seek_pattern()

if the beginning of the line matches a pattern which is being ignored, it will skip it and look at the next line

gpg_binary = 'gpg'

options = {'command-fd': 0, 'fixed-list-mode': None, 'with-fingerprint': None, 'list-options': 'show-sig-subpackets,sho

seek (*fd*, *pattern*)
look for a specific GNUPG status line in the output

this is a stub for seek_pattern()

seek_pattern (*fd*, *pattern*)
iterate over file descriptor until certain pattern is found

fd is a file descriptor pattern a string describing a regular expression to match

this will skip lines not matching pattern until the pattern is found. it will raise an IOError if the pattern is not found and EOF is reached.

this may hang for streams that do not send EOF or are waiting for input.

set_option (*option*, *value=None*)
set an option to pass to gpg

this adds the given 'option' commandline argument with the value 'value'. to pass a flag without an argument, use 'None' for value

unset_option (*option*)
remove an option from the gpg commandline

version ()
return the version of the GPG binary

write (*fd*, *message*)
write the specified message to gnupg, usually on stdout

but really, the pipes are often setup outside of here so the fd is hardcoded here

exception monkeysign.gpg.**GpgProtocolError**
simple exception raised when we have trouble talking with GPG

we try to pass the subprocess.popen.returncode as an errorno and a significant description string

this error shouldn't be propagated to the user, because it will contain mostly "expect" jargon from the DETAILS.txt file. the gpg module should instead raise a GpgRutimeError with a user-readable error message (e.g. "key not found").

expected ()

found ()

match ()

exception monkeysign.gpg.**GpgRuntimeError**

class monkeysign.gpg.**Keyring** (*homedir=None*)
Keyring functionalities.

This allows various operations (e.g. listing, signing, exporting data) on a keyring.

Concretely, we talk about a "keyring", but we really mean a set of public and private keyrings and their trust databases. In practice, this is the equivalent of the GNUPGHOME or –homedir in GPG, and in fact this is implemented by setting a specific homedir to tell GPG to operate on a specific keyring.

We actually use the –homedir parameter to gpg to set the keyring we operate upon.

context = None

decrypt_data (*data*)
decrypt data using asymetric encryption

returns the plaintext data or raise a GpgRuntimeError if it failed.

del_uid(*fingerprint*, *pattern*)

encrypt_data(*data*, *recipient*)
encrypt data using asymetric encryption

returns the encrypted data or raise a GpgRuntimeError if it fails

export_data(*fpr=None*, *secret=False*)
Export OpenPGP data blocks from the keyring.

This exports actual OpenPGP data, by default in binary format, but can also be exported asci-armored by setting the 'armor' option.

fetch_keys(*fpr*, *keyserver=None*)
Download keys from a keyserver into the local keyring

This expects a fingerprint (or a at least a key id).

Returns true if the command succeeded.

get_agent_socket()
get the location of the gpg-agent socket for this keyring

get_keys(*pattern=None*, *secret=False*, *public=True*, *keys=None*)
load keys matching a specific patterns

this uses the (rather poor) list-keys API to load keys information

import_data(*data*)
Import OpenPGP data blocks into the keyring.

This takes actual OpenPGP data, ascii-armored or not, gpg will gladly take it. This can be signatures, public, private keys, etc.

You may need to set import-flags to import non-exportable signatures, however.

sign_key(*pattern*, *signall=False*, *local=False*)
sign a OpenPGP public key

By default it looks up and signs a specific uid, but it can also sign all uids in one shot thanks to GPG's optimization on that.

The pattern here should be a full user id if we sign a specific key (default) or any pattern (fingerprint, keyid, partial user id) that GPG will accept if we sign all uids.

@todo that this currently block if the pattern specifies an incomplete UID and we do not sign all keys.

verify_file(*sigfile*, *filename*)

class monkeysign.gpg.**OpenPGPkey**(*data=None*)
An OpenPGP key.

Some of this datastructure is taken verbatim from GPGME.

algo = -1

creation = 0

disabled = False

expired

expiry
Returns a datetime from the _expiry field or None if the key does not expire

format_fpr()
display a clean version of the fingerprint

this is the display we usually see

fpr = None

get_trust()

invalid = False

keyid(*l=8*)

length = None

parse_gpg_list(*text*)

purpose = {}

qualified = False

revoked
Returns whether GnuPG thinks the key has been revoked

This is the second field of the result of the –list-key –with-colons call. Note that this information is only present on public keys, i.e. not on secret keys.

Returns None if it cannot be determined whether this key has been revoked.

secret = False

subkeys = {}

trust = None

trust_map = {'': 'empty', '-': 'unknown', 'e': 'expired', 'd': 'disabled', 'f': 'full', 'i': 'invalid', 'm': 'marginal', 'o': 'ne

uids = {}

class monkeysign.gpg.**OpenPGPuid**(*uid*, *trust*, *creation=0*, *expire=None*, *uidhash=''*)

get_trust()

revoked
Whether this UID has been revoked

Note that, due to GnuPG not exporting that information for secret keys, UIDs of secret keys do not carry that information.

Return None if it cannot be determined whether this UID has been revoked. Try again with the public key.

class monkeysign.gpg.**TempKeyring**

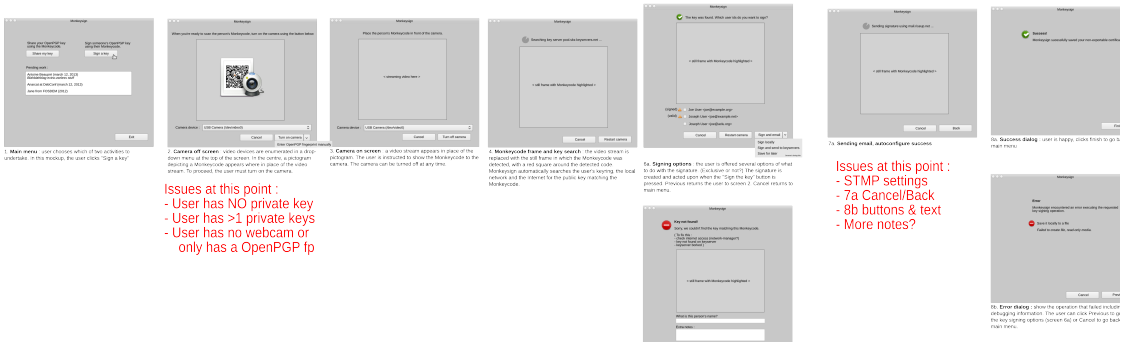
CLI Interface

GTK Interface

UI mockups and design

We are planning significant changes to the graphical user interface of Monkeysign in the 3.x branch.

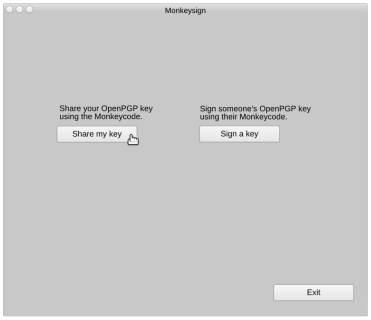
Signing interface



Issues at this point :
- User has NO private key
- User has >1 private keys
- User has no webcam or only has a OpenPGP fp

Issues at this point :
- STMP settings
- 7a Cancel/Back
- 8b buttons & text
- More notes?

Key sharing interface



1. **Main menu** : user chooses which of two activities to undertake. In this mockup, the user clicks "Share my key"

3a. **Monkeycode** : the Monkeycode is generated and displayed on the screen, with a hint as to how to use the code. The "Options" section is collapsed. The "change identity" dropdown is showed only if multiple secret keys are available.

3b. **Monkeycode** : the user can expand the Other options section by clicking on the title. If there is only one valid key, we display a label. The text (in the label or dropdown) must be unambiguous: if there are many keys, show the creation date. Show "expired" if it is.

4. **Finish** : After having the Monkeycode scanned, or printed or saved as an image, the user clicks Finish and is returned to screen 1.

Terminology

In this documentation, the following definitions are used:

QR-code

QRcode

QR code

Bar code

Barcode A "QR code" (abbreviation of Quick Response Code) is sort of bar code, an optical label that is designed to be machine-readable. A QR code is faster to read by computers and contains more information than than regular bar codes, which is why it is used in Monkeysign to communicate fingerprints. See Wikipedia article QR code for more information.

MTA

Message Transfer Agent A computer program designed to transfer emails between different machines, usually running on servers. See Wikipedia article Message Transfer Agent for more information.

MUA

Mail User Agent A computer program used to read, compose and send email, normally ran on user computers. See Wikipedia article Mail User Agent for more information.

We also try to adhere to the Modern PGP terminology when possible.

Credits

Those people are the ones who made Monkeysign possible.

```
__authors__ = ['In alphabetical order:',
               '',
               'Antoine Beaupré',
               'Daniel Kahn Gillmor',
               'Gabriel Fillion',
               'Jérôme Charaoui',
               'Kristian Fiskerstrand',
               'Philip Jägenstedt',
               'Ramakrishnan Muthukrishnan',
               'Simon Fondrie-Teitler',
               'Tobias Mueller',
               ]
__documenters__ = ['In alphabetical order:',
                   '',
                   'Antoine Beaupré',
                   'Emma Peel',
                   ]
__translators__ = ['In alphabetical order:',
                   '',
                   'Antoine Beaupré',
                   'Ahmed El Azzabi',
                   'Gonzalo Exequiel Pedone',
                   'Michael R. Lawrence',
                   'Michal Čihař',
                   ]
```

- genindex
- modindex
- search

Python Module Index

m

Index

R

S

T

U

V

W