How to use pgp

This article is to help release managers create pgp keys for signing.

Who Needs to Sign?

If you are a release manager, you will need to digitally sign the release tarball.

Where can I learn more?

These instructions are derived from the apache guide on release signing. You'll need to know how to do the steps presented in this guide if you are a release manager.

- A good overview of why we sign is here. It explains the concept of *web of trust*.
- An excellent example of another Apache project's release guide is here.

Why does this seem so complicated?

Learning how to use public/private key crypto is not as simple as learning how to send an email. Granted, it's also not as hard as learning c++, so there's reason to be thankful.

I think a large part of why this process seems complicated is that you must be familiar with a fairly technical concept and at the same time learn yet another command line tool. This meaning you'll have to learn about asymetric cryptography as well as the tool which implements the algorithms, either pgp, gpg, or kpgp.

Keep hope, and do please note that, as per the apache guide,

> *Applied cryptography is a subject that has considerable depth. Luckily, it's possible to get started signing releases without being an expert. Just remember that (from time to time) you will encounter situations that will require research and learning. Hopefully the FAQ will be a reasonable first port of call.*

How do I sign a release?

Generate a key

If you already have a key pair created with pgp and it's registered on a public keyserver, then you can skip this step. If not, here's how to generate and upload a key which is linked to you (via an email address).

First, here's the link to the apache page which these instructions are derived from.

Next, here's one possible set of scripts that you can use to generate your key on a mac.

Get gpg, the GNUpgp implementation. sh # Installing gpg on a Mac

brew install gpg # Of course, this assumes you have brew.

Next, change the settings of `gpg` so that a strong security standard is the default. Of course, they may already be default, so you'll want to check if your gpg.conf file specifies this.

```
# Appending some settings to ~/.gnupg/gpg.conf

# gpg must run at least once to create its config folders.
gpg --version
echo "personal-digest-preferences SHA512" >> ~/.gnupg/gpg.conf
echo "cert-digest-algo SHA512" >> ~/.gnupg/gpg.conf
echo "default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP Uncompressed" >>
~/.gnupg/gpg.conf
```

Now you can generate a new key using the strong security settings. Note that you should accept the defaults, and non-expiring key option.

```
# Generating a public/private key pair

gpg --gen-key
```

Once that's done, you'll want to register your new public key with a keyserver. This will make it easier for resistance operatives to help you take down tyrannical dictatorships. More generally, they help establish your identity but in no way guarantee it. You can have other people in the Apache web of trust sign your key so that it is more trusted.

```
# Finding the id of your new key

gpg --list-keys
# pub   2048R/D64EA123 2017-01-17
# uid       [ultimate] Marc Spehlmann (EXAMPLE) <ajmarc@cramja.com>
# 'D64EA123' is the key's id

gpg --send-key YOUR_KEY_ID
# ex: gpg --send-key D64EA123

# there is also a keyserver option to choose which keyserver
#  gpg --keyserver keyserver.ubuntu.com --send-key D64EA123
```

Exporting your key to KEYS

You'll want to export the public key from the public/private key pair you are going to use to sign the release. We keep this in a KEYS file in the root directory of Quickstep, following the example of other Apache projects.

```
# Appending your public key to the KEYS file

cd ~/workspace/incubator-quickstep
gpg --export --armor YOUR_KEY_ID >> KEYS
# ex: gpg --export --armor D64EA123 >> KEYS
```

Signing the Release Tarball

We sign the release so that a downloader can verify that what they downloaded is identical to the object which you signed. This is for security.

Checksums have the same effect, but are more generally to detect errors in transmission.

```
# Signing a release

gpg -u YOUR_APACHE_USER_NAME@apache.org --armor --output apache-quickstep-incubating-x.y.z.tar.gz.asc --detach-
sign apache-quickstep-incubating-x.y.z.tar.gz

# Make sure it worked
gpg --verify apache-quickstep-incubating-x.y.z.tar.gz.asc apache-quickstep-incubating-x.y.z.tar.gz

# Make checksums
md5sum apache-quickstep-incubating-x.y.z.tar.gz > apache-quickstep-incubating-x.y.z.tar.gz.md5
sha1sum apache-quickstep-incubating-x.y.z.tar.gz > apache-quickstep-incubating-x.y.z.tar.gz.sha

# Make sure they worked
md5sum --check apache-quickstep-incubating-x.y.z.tar.gz.md5
sha1sum --check apache-quickstep-incubating-x.y.z.tar.gz.sha
```

Signing another Committer's key

```
gpg --list-keys
# now copy your apache key ID.
# Let's pretend that you're verifying Marc's key. This means you have already added Marc's public key
gpg -u YOUR_KEYID --sign-key spehl@apache.org
# Now send Marc's updated information
# notice that 5A29899A is Marc's key ID
gpg --keyserver keyserver.ubuntu.com --send-key 5A29899A
# you have now verified that Marc's key is from Marc
# The --keyserver is optional, but then you must have a valid default
```

Related articles

Content by label

There is no content with the specified labels