Using GPG and Keybase to sign files, etc.

June 9, 2018

Thought about using this to sign releases in github.

Here are some resources: (And other stuff that I learned.)

From Linuxmint website.

| **How to verify ISO images** This page explains how to verify their integrity and authenticity. **Preparation** 1. Create a directory called "ISO" in your home directory.  2. Move the ISO image you downloaded in this directory.  3. Download the following files and move them into the "ISO" directory.   | **FILE** | **DESCRIPTION** | | --- | --- | | [sha256sum.txt](https://ftp.heanet.ie/mirrors/linuxmint.com/stable/18.3/sha256sum.txt) | Contains the SHA256 sums to check the integrity of the ISO images. | | [sha256sum.txt.gpg](https://ftp.heanet.ie/mirrors/linuxmint.com/stable/18.3/sha256sum.txt.gpg) | Signed by the Linux Mint team to check the authenticity of the sha256sum.txt file. |   Your ~/ISO directory should now contain 3 files: Your ISO image, the sha256sum.txt file and the sha256sum.txt.gpg file. **Integrity check** To verify the integrity of your ISO image, generate its SHA256 sum and compare it to the one found in the sha256sum.txt file.  In most Linux distributions the SHA256 sum can be generated by opening a terminal and running the following commands:  **cd cd ISO sha256sum -b \*.iso**  The last command should show you the SHA256 sum of your ISO file. Compare it to the one found in the sha256sum.txt. If they match, you've successfully verified the integrity of your ISO image.  Note: If you have coreutils version 8.25 or newer, another way of checking the sum is to ask the sha256sum command to check the file against the sha256sum.txt file, like this:  **sha256sum --ignore-missing -c sha256sum.txt** **Authenticity check** To verify the authenticity of the sha256sum.txt file, we need to check the signature on the sha256sum.txt.gpg file.  1. Import the Linux Mint signing key:  **gpg --keyserver keyserver.ubuntu.com --recv-key "27DE B156 44C6 B3CF 3BD7 D291 300F 846B A25B AE09"**    Note: If gpg complains about the key ID, try the following commands instead:  **gpg --keyserver keyserver.ubuntu.com --recv-key A25BAE09**    **gpg --list-key --with-fingerprint A25BAE09**    Check the output of the last command, to make sure the fingerprint is 27DE B156 44C6 B3CF 3BD7 D291 300F 846B A25B AE09.  2. Verify the authenticity of the sha256sum.txt file:  **cd cd ISO gpg --verify sha256sum.txt.gpg sha256sum.txt**  The output of the last command should tell you that the file signature is 'good' and that it was signed with the following key: **A25BAE09**.  Note: Unless you trusted this signature in the past, or a signature which trusted it, GPG should warn you that the signature is not trusted. This is expected and perfectly normal. |
| --- | --- | --- | --- | --- | --- | --- |

<https://keybase.io> - encrypted end to end chat/file share/git/and teams

Making a signed gpg file with detached signature.

<https://security.stackexchange.com/questions/104149/make-signed-file-from-content-file-and-its-detached-signature>

Creating a signed Github release

<https://wiki.debian.org/Creating%20signed%20GitHub%20releases>

My public GPG key:

<https://pgp.mit.edu/pks/lookup?op=get&search=0xB6B7DA615E5C9A9D>

Exchanging GPG keys:

<https://www.gnupg.org/gph/en/manual/x56.html>

(Basics:

Download the public key file, on the command line type gpg --import publickeyfilename.gpg

Verify key with gpg --fingerprint emailaddress

Or gpg --finerprint “leroy miller”

) My fingerprint should be: 9545 12FF F306 04E0 F6FB E4BB B6B7 DA61 5E5C 9A9D

If you get something else than you have a invalid key.

We can also import the key from a pubic keyserver like this:

gpg --keyserver pgp.mit.edu --recv-keys 5E5C9A9D

# To sign a file with gpg and detached signature:

First we need to zip or tar the file and create a sha256sum file - this is what we will be comparing later.

sha256sum -b filename.zip >sha256sum.txt

Next we will create a gpg signature for that file. And Since more likely than not we are going to publish this signature on a web site we need to use the gpg --armor feature.

gpg --armor --output sha256sum.txt.sig --detach-sign sha256sum.txt

Now, we publish the original zip file, the sha256sum.txt file, and the sha256sum.txt.sig file. And we can then verify that we have a good download by generating a sha256sum of the file we downloaded and compairing to the contents of the sha256sum.txt file, we can verify the sha256sum.txt file, by the signature.

So lets say we downloaded a zip file: filename.zip

We can verify it one of two ways:

Long hard way -

sha256sum filename.zip

Which will return something like:

6157770e393822993ac6406797d8cb3872c1d07a9e1127badcc6cc963a84fab4 filename.zip

cat sha256sum.txt

Which should show the same as the above, and you can verify manually that it does.

Or we can just -

sha256sum -c sha256sum.txt

Which should say

Filename.zip: OK

If everything is good or

filename.zip: FAILED

If there is a problem.

To verify that the sha256sum.txt is legit -

We use gpg

gpg --verify sha256sum.txt.sig sha256sum.txt

IF everything is good you should get something that looks like this:

gpg: Signature made Sat 09 Jun 2018 08:28:42 PM EDT using RSA key ID 5E5C9A9D

gpg: Good signature from "LeRoy Miller <kxxxxp@xxxxxxx.com>"

gpg: aka "LeRoy Miller <lxxxxxxx@xxxxxxxx.com>"

gpg: aka "kxxxxxp@xxxxxxx.com"

IF the file has been messed with you will get something like this:

gpg: Signature made Sat 09 Jun 2018 08:28:42 PM EDT using RSA key ID 5E5C9A9D

gpg: BAD signature from "LeRoy Miller <kxxxxxxxx@xxxxxxxl.com>"

If you download all three files from the same source good chance everything will pass.

IF you download the main file (filename.zip) from a different source, it’s a good way to verify and catch a problem before it gets ot be a problem.

# GPG Sign a message in the clear (not encrypted)

First we write and save the message as a text file.

We are going to call it message.txt

| Hello World. |
| --- |

Now we need to sign it:

gpg -a --clearsign message.txt

We will get a file called:

Message.txt.asc

Which will look something like this:

| -----BEGIN PGP SIGNED MESSAGE-----  Hash: SHA1  Hello World.  -----BEGIN PGP SIGNATURE-----  Version: GnuPG v1  iQEcBAEBAgAGBQJbHJA5AAoJELa32mFeXJqdPr8H/0ZhbKS1P4C3q/ED+Fa0kpF+  XqNOQlxN6r5YjrXktP3nT+gKJ0wi88QjkMdn2Cqaszm+2j/6YBbNeCbrIEq3f16O  zcVs2Yd78483SS01L72Foe7S8Zf3sq+1QCkvUpTOmajrAA1pbKM2sqNXmSnDyNef  kI3cBcumm4cw7iZ2KnNuMX7qXcuXdV6iDCKJ2wuf+x7R4eV7tZqwuv89etkwXHO4  Re8yk59APJcvb33fAm8zhu4ROpXEWUqpynuVyLUA918XWk9kSzk1M+BXCemqIFXn  r3pb1ecTxXY15JaA/ew7NwQ5oCJbDLs5+RnVO0wM84uTHteDAi6A9a50cVH3bog=  =9EwQ  -----END PGP SIGNATURE----- |
| --- |

You can see the message “Hello World.” is in the clear, so it’s readable by everyone.

We can now verify the signature - as long as we have the public key in our keyring.

gpg --verify message.txt.asc

And you should see a good signature.

You may also get a warning if you are doing this yourself, and trying to test this out.

You may get a WARNING not a detached signature; file ‘message.txt’ was NOT verified.

(IF you rename or remove message.txt at this point, and leave message.txt.asc, this warning will go away.)

I found this comment on github explaining the problem and issue

<https://github.com/nodejs/node/issues/6821> (They are talking about a different file, but the problem is the same)

| [rvagg](https://github.com/rvagg) **commented [on May 18, 2016](https://github.com/nodejs/node/issues/6821#issuecomment-220033176)**  | OK, so here's the thing that's going on here: when you do a --verify, gpg will look at the file being verified and if there's an equivalent non-signed file then it'll assume you're working with a detached signature, which we don't do. This is happening here because you're also downloading SHASUMS256.txtand have it beside SHASUMS256.txt.asc. Try removing or renaming the former and you'll get a different result because gpg will decide it's a cleartext signed document with the sig *inside* the doc rather than detached from it.  But this raises an interesting point because detached signatures offer a bit more safety than we offer and maybe we should switch our signing mechanism to use them instead, or as well. The reason is that gpgwill only verify the contents between -----BEGIN PGP SIGNED MESSAGE----- and -----BEGIN PGP SIGNATURE----- using the signature found within. But we are recommending a simple grep. So someone wishing to insert a bad build onto nodejs.org could just add a new line outside of the validated block with their new shasum and the filename and the file would still get verified and the invalid shasum would be grepped just fine. i.e. we don't have enough steps on our README to properly handle this case so it's actually not all that secure.  Detached signatures give you a signature for an entire file in a separate file. If we shipped a SHASUMS256.txt.sig as a detached signature then you'd download that as well as SHASUMS256.txt and gpg --verify would check the sig against the original and verify the whole thing. Then our instructions on the README about using grep against SHASUMS256.txt would be perfectly acceptable.  [@nodejs/crypto](https://github.com/orgs/nodejs/teams/crypto) [@jbergstroem](https://github.com/jbergstroem) can you --verify my logic above ^? | | --- | |
| --- | --- |

# Keybase.io

Keybase is even more powerful than gpg, but isn’t as widely used yet. It’s focus on more on encrypted end to end chat, teams and file sharing. It also can encrypt a git. You can generate encrypted messages and even sign files using a method similar to above.

The steps to do it are about the same, take a look at the keybase.io website for more.

# Sign a message with Keybase

We can also sign a message with keybase, but the message is hashed out. The process is similar to using gpg We write a message in the clear, and save it.

Message.txt our “Hello World.” message for example.

We then use keybase to sign it.

Keybase sign -i message.txt -o message.txt.keybase

We will now get a file that looks like this:

| BEGIN KEYBASE SALTPACK SIGNED MESSAGE. kXR7VktZdyH7rvq v5weRa0zkXrVVui DCPwskqGlqtL1Ue Utaq8Kqykbm1qXM e57f3dL1cdYuJxB WBULkBRLgtai0nV 2SU7X267wfpXGgF 07bhBwbbX7dDSXj XeG0XdINPQyYsG5 DxpQ6OFQmRteAF7 gvUhsm3IBXzVZQT YPqfEP8abzx92vw ZVK0tYEO2PwGi6j SNQ9c0Ta. END KEYBASE SALTPACK SIGNED MESSAGE. |
| --- |

To verify the message we do this:

keybase verify -i message.txt.keybase

And we get this:

| Signature verified. Signed by kd8bxp (you).  Hello World. |
| --- |

Which as you can see, this was verified, and it was signed by me - and the message is Hello World.