# Allan McRae

I love gluten




# Two PGP Keyrings for Package Management in Arch Linux

Posted in **Arch Linux**, **Pacman** on 2015/01/03 by Allan

Both the `pacman` package manager and the `makepkg` tool for building packages verify files using PGP signatures. However, these two pieces of software do it using different keyrings. There seems to be a lot of confusion about this and misinformation is spreading at a rapid pace, so I'll attempt to clarify it here!

**Pacman Package File Signature Verification**

By default, pacman is set-up to verify every package using a PGP signature. It has its own keychain for this purpose, located at `/etc/pacman.d/gnupg/`. This keychain is initialized during the Arch Linux install – a root key is created and the Arch Linux master keys are locally signed by the root key. The master keys sign all Arch Developer and Trusted User keys, creating an effective web-of-trust from your pacman root key to each of the packager keys allowing verification of package files.

If you want to allow the installation of package files from a non-official repository, you need to either disable signature verification (don't do that…), or trust the packagers signing key. To do this you first need to verify their key ID, which should be well publicized. Then you import it into the pacman keyring using "`pacman-key --recv-key <KEYID>`" and signify that you trust the key by locally signing it with your pamcan root key by running "`pacman-key --lsign <KEYID>`".

**Makepkg Source File Signature Verification**

When building a package, the source files are often (and should be!) signed, with a signature file available for download alongside the source file. This typically has the same name as the source file with the extension `.sig` or `.asc`.`makepkg` will automatically verify the signature if it is downloaded in the sources array. e.g.:

```
source=(http://ftp.gnu.org/gnu/libc/${pkgname}-${pkgver}.tar.xz{,.sig})
```

However, `makepkg` needs some information to verify the source signature. It will need the public PGP key of the person who signed the source file, and that key to be trusted. The difference here is that you do not trust whoever provided the source file to provide packages for your system (or at least you should not the vast majority of the time), so your user's keyring is used. To get the key use "`gpg --recv-key <KEYID>`" and trust it (once suitably verified) using "`gpg --lsign <KEYID>`".

If you provide a package to the AUR, it would be a lot of work for everyone to suitably verify a PGP key and locally sign it. To demonstrate that you have verified the key, you can add the following to the PKGBUILD:

```
validpgpkeys=('F37CDAB708E65EA183FD1AF625EF0A436C2A4AFF') # Carlos O'Donell
```

Now `makepkg` will trust that key, even if it is not trusted in the package builder's PGP keyring. The builder will still need to download the key, but that can be automated in their gpg.conf file.

Hopefully that clarifies the two separate types of PGP signature verification happening in `pacman` and `makepkg` and explains why they should be separate… Now can people stop recommending that the `pacman` keyring is imported into the user's keyring and *vice versa*?

This entry was posted in **Arch Linux**, **Pacman** by **Allan**. Bookmark the **permalink [http://allanmcrae.com/2015/01/two-pgp-keyrings-for-package-management-in-arch-linux/]** .

---

**16 thoughts on "TWO PGP KEYRINGS FOR PACKAGE MANAGEMENT IN ARCH LINUX"**

**Johannes Löthberg**
on **2015/01/03 at 12:10 PM** said:

"I approve of this message."

(Now please forgive me, Oh Great One.)

**Christian Hesse**
on **2015/01/04 at 8:05 AM** said:

The main problem still persists: Users think that "FAILED (unknown public key XXX)" is packager's fault. (Some AUR packages get flooded with comments about that.) makepkg should give a hint like "Please import public key to your keyring, then try again."

Anonymous
on **2015/01/04 at 6:04 PM** said:

"Hopefully that clarifies the two separate types of PGP signature verification happening in pacman and makepkg and explains why they should be separate… Now can people stop recommending that the pacman keyring is imported into the user's keyring and vice versa?"

Fine, but why not already have makepkg use what's already in pacman's keyring? I don't understand your explanation. All this seems to do is make more steps to installing packages from the official repos via the ABS. I get not automatically accepting keys from the AUR, but this doesn't really clarify for me why makepkg can't already go with what's already been trusted.

All this change did was disrupt my workflow and annoy me because suddenly makepkg wasn't doing something it was doing fine BEFORE the 4.2 update.

Allan

on **2015/01/04 at 6:18 PM** said:

People who provide Arch Linux packages and people who develop software that you want to make a package from are two completely separate sets of people. Even using the keys in pacman's keyring will not get you very far (apart from software that Arch devs are in charge of – e.g. cower, which I think is the cause of all this confusion).

With pacman-4.2, makepkg when from verifying the PGP signatures of source tarballs but ignoring all but the worst failures, to taking note of all failures. You can ignore the PGP signature checks using –skippgpchecks, but that is a bad idea. Checksums are not a great way to verify a tarballs validity, especially when they are probably provided by the packager and not the upstream developer.

Rob Hasselbaum

on **2015/01/05 at 1:39 AM** said:

Thanks for the background. I ran across this post while trying to solve a key signing problem for cower in the AUR. But unfortunately, your advice directly contradicts the package maintainer, who says in the AUR comments "Do NOT locally sign my key." I assume that's because the PKGBUILD has a validpgpkeys entry, so then it's not necessary to locally sign the key in those cases…?

Allan

on **2015/01/05 at 8:48 AM** said:

It is not necessary to lsign a key if it is in validpgpkeys. It is also not be the packagers key – it is the software developers.

Ron

on **2015/01/05 at 7:12 AM** said:

To get the key use “gpg –recv-key ” and trust it (once suitably verified) using “gpg –lsign “.

Could you give us a little more information about getting and verifying s? Say I use an (orphaned, perhaps) AUR package (such as passwordsafe, just to name an example), which I cannot build at the moment (using yaourt, in this example). I have the sig file. So, firrst question: How do who signed it? And (2nd question), how do I go about verifying this? Sorry if that’s clear to everyone else, but it’s not clear to me (and, since in this example we’re talking about a cryptographic application that I would like to be somewhat paranoid about, I would like to understand this fairly completely).

Thanks in advance!

Allan

on **2015/01/05 at 9:30 AM** said:

If you have a sig file, you can try verifying the tarball with “gpg –verify tarball.sig”. That will give the key ID if it is not in your keyring. Once you have the key ID, you can use “gpg –recv-key ” to get the key, and “gpg –fingerprint ” to see the details.

I then check the person who signed is expected from the software mailing list, check if they sign emails, look if the PGP fingerprint is published on their homepage, … That verifies the signature enough to add the validpgpkeys array for me.

Ron

on **2015/01/05 at 7:35 PM** said:

Thanks! That allowed me to get the package built. Bit uncertain about the verification, but that’s par for the course, I guess.

By the way, it might help others to note that, when adding a “validpgpkeys” line to the PKGBUILD, as deccribed in the original post, you use the “Key Fingerprint”, which you obtain when running “gpg –fingerprint KEYID”. Oh, and apparently you can’t just cut and paste – it didn’t work for me until I removed the white space, as in your example.

Ron

on **2015/01/05 at 7:16 AM** said:

Postscript: For some reason, your blog software removes word in all caps, apparently. Or maybe not, looking at the other posts – but everywhere I wrote “Key Id” in all caps, it’s disappeared from my post. The first line was a quote from your posting, you can see what I mean if you compare. In any case, my questions were (quite simply): How do I get the key id, given the sig file, and how do I verify it (generally speaking). Sorry if my comment was confusing… Thanks.

cippaciong

on **2015/01/05 at 10:39 PM** said:

I think this article should have more visibility among Arch users, a lot of them are having troubles on AUR with PGP and the problem is not circumscribed to cower (see owncloud-client). Why don’t you create a new entry in Archlinux News pointing to this page or an equivalent topic on the Arch forum?

Claire Farron

on **2015/01/11 at 10:04 PM** said:

I support this, as I’m starting to get flooded with e-mails saying I’ve broken my PKGBUILDs because people aren’t aware of the change.

Jristz

on **2015/01/06 at 4:43 AM** said:

Is pacman/makepkg tiedo to gnupg or I can replace it with another software (if it provide the same binaries)?

**Anonymous**

on **2015/01/27 at 6:29 PM** said:

All wet because:

(1) A user’s keyring is for user e-mail, not AUR cruft. That is system stuff.

(2) Sensitive security stuff held in a user account is a breach.
(3) One who wants AURs must know GPG tech…not your typical user.
(4) The claim that we should trust Arch builds more than our own is bogus. One must verify a binary or tarball, either way. MITM attacks are equally possible either way.
(5) It's just an Arch dev team convenience. Arch devs write each other, and upstream, while dev'ing packages, which they must sign. They're a corner case.

Anyone else should factor out pacman cruft from personal keyrings. Fetch them like so (or you can use a hex fingerprint search field),

wget -O vdpauinfo.asc "http://pool.sks-keyservers.net:11371/pks/lookup?op=get&search=0x1BEF3D8401A68861"

In the download file, strip XML, then gpg –dearmor into /usr/share/pacman/keyrings. Finish with pacman-key –populate.

Yet…yet…I don't see why we should be tasked at all.

1) Arch itself might add packages for keyrings – Debian's, tor's, etc. Any AUR could then dep itself to a keyring from Arch proper, which Arch devs would sign. ZERO USER INVOLVEMENT.

2) Any AUR pkg maintainer could, right now, on his own, define a separate AUR pkg to plop a public key in /usr/share/pacman/keyrings. The main AUR pkg would then dep on the AUR keyring pkg. Here, the maintainer tracks keys for ALL users, turning the work of N individuals into the work of 1 individual. There would be lots less room for error, lots less work, lots less forum madness and blogging.

I'm sure many can improve my ideas; I just refute the concept that a user keyring ought to handle packaging tasks.

Allan

on **2015/01/27 at 7:01 PM** said:

Did you read at all why it is a very, very bad idea to put keys for source tarballs in you pacman keyring? No keys should go in that keyring beyond those of people you trust to have signed a package you are installing. This is equivalent to the Debian keyring, which only has Debian developer/maintainer keys in it.

If you want a separate keyring for packaging, use the GNUPGHOME variable when calling makepkg. Even stick it in makepkg.conf. But I fail to see how using the users keyring is a "breach". The user is building the package. The administrator is installing packages which are verified with a completely different keyring.

whaler

on **2015/02/18 at 12:50 AM** said:

Well, I am probably not the brightest. Trying to upgrade iucode-tool I have tried $ “gpg –recv-keys 123….”, but get “gpg: input from key server failed: No keyserver available”. I am typing this on the command line while sudo’ed (after using Yaourt).

I have also created
“~/.gnupg/gpg.conf” with
“keyserver-options auto-key-retrieve”
as advised in the Wiki, to no avail.

Should I specify a key server somewhere? What am I overlooking or doing wrong?