GPG In Emacs

Apr 10, 2020 • emacs cryptography • 3 comments

[Table of Contents]

I recently decided to keep a private journal. I wanted my journaling to be efficient and comfortable, so of course I've been doing it in Emacs. Even though I do minimal editing, writing in a stream-of-consciousness style, I feel at home in Emacs.

I also wanted journal entries to be private, so I thought I'd encrypt them. I didn't want to risk getting hacked and my dark inner secrets spilling out into the world. I also use Dropbox to keep a cloud copy of all my data, and I wanted to stuff my journal in there for convenience. This meant I had extra reason to secure it with encryption.

As Emacs users, our best choice for editing encrypted files is EasyPG. It's not the greatest or the most intuitive tool, and it's based on GnuPG which is itself a nightmare of UX design. However, after some annoying initial setup, it is convenient. This is a guide to that annoying initial setup.

What is EasyPG?

EasyPG is like cherry ice cream in a cone[1], in the sense that it lets you access encrypted data transparently. When you visit a file such as `secret.txt.gpg`, EasyPG – or `epa` – yeah, I don't get the acronym, either – will automatically decrypt it and open it as it it were a regular text file. Spiffy, eh? When you save, bada bing, bada boom: `epa` encrypts it for you automatically!

The backbone of EasyPG is GnuPG. Expanding on the dessert analogy, GnuPG is like an oatmeal raisin cookie. Enough said, right?

What's GnuPG?

Read the wiki page.

Generating your GPG keys

The first step to using EasyPG is to set up GnuPG, which is its backbone. Now, what you want to do at this stage is decide if it's worth the hassle. It's not like some cool hacker like me wouldn't be able to pwn you, anyway.

Okay, what you want to do now is generate your GPG keys. Hold it right there! Make sure you have `gnupg` installed before you take another step. On OSX, you'll want to use Homebrew, and then type out `brew install gnupg` in eshell or shell or fish or zsh. GnuPG is the backbone of EasyPG.

> **Note:**
> Make sure you're using a recent `gnupg` version – I'm on `2.2.17` at the time of this writing. My `epa` setup (see below) will probably not work with `gnupg 2.0`, and it will *definitely* not work with `< 2.0`.

So, generating your GPG key. What you wanna do here is type out the following.

```
gpg --full-generate-key
```

Now do this:

1. Select the default key kind.
2. Select the keysize you want. Higher is more secure.
3. Make a *never-expiring key*.
4. Type in your real, full name and email.
5. Type in a password.

And bada bing, bada boom you've generated a key in the `~/.gnupg/` directory!

> **Note:**
> **Be very cautious here not to make any mistakes. This is a serious program, made for serious users. It is NOT for kids.**

Note:

Remember your password! **This is the password you will use when decrypting with `epa`.**

Note:

When picking a password, length counts. I'm a fan of the diceware technique for generating long, yet easy-to-remember passwords. (With my `epa` setup (see later), `epa` will cache your password for a while; you don't have to type it in every time.)

Securing your GPG key

This is an important point: you **mustn't** lose the generated key in `~/.gnupg/`. **Both this key and the password will be required to decrypt your files.** What I did personally is I moved this directory to Dropbox and created a symlink to it in my home directory[2]. I regularly make backups of my Dropbox folder. This way the key is stored on my computer, in the cloud and on my own external disks.

Setting up EasyPG

Okay, great, we got past the part with GnuPG. We'll never have to touch this program again. You did create never-expiring keys, right?

Now let's set up EasyPG. EasyPG is a built-in Emacs package. If you're not using Emacs, you just wasted your time reading this post. Thanks for the page hit. Sucker.

Let's set up EasyPG. Here is the config I put in my Emacs init.

```
;; Don't bring up key recipient dialogue.
(require 'epa-file)
(setq epa-file-select-keys nil)
(setq epa-file-encrypt-to '("<YOUR EMAIL HERE>"))
```

This first snippet makes it so you don't have to provide your email address every time, I think. And something about a "key recipient dialogue"? I have no clue.

about a "key recipient dialogue"? I have no clue.

Optionally, you may wish to increase the password expiry time to something longer and a bit more convenient, like 15 minutes.

```
;; Increase the password cache expiry time.
(setq password-cache-expiry (* 60 15))
```

I'm not actually sure if this does anything, since half of these settings seem to be deprecated. Whatever.

> **Update:**
> This variable has no effect if you use GPG2, which automatically runs `gpg-agent` to cache the password. You'll have to set one of the options like `default-cache-ttl` in `~/.gnupg/gpg-agent.conf`.

This next part might not be needed for you. I think it depends on your Emacs version. If you get an error like `inappropriate ioctl for device`, please include this fix (from here):

```
;; Fix EasyPG error.
;; From https://colinxy.github.io/software-installation/2016/09/24/emacs25-easypg-issue.html.
(defvar epa-pinentry-mode)
(setq epa-pinentry-mode 'loopback)
```

I have no idea what this is about or how it works. All I can say is that it is absolutely pathetic that EasyPG is broken out of the box. As time goes on, I become less and less a fan of open source software and the largely talent-less developers behind it. All this time and effort, setting up something that should just work, just because I didn't want to shell out the 80 buckaroos for Sublime Text. Wait, Emacs is awesome. Sucker.

That's pretty much it, I guess. The EasyPG setup is pretty finicky and I honestly just cobbled together something that seems to work fine. Who cares.

Let's go for a test drive!!!

1. Open up Emacs and create a file called `test.txt.gpg`.
2. Enter some arbitrary text, like `Vim and Sublime are inferior text editors for total suckers.`
3. Save it. A dialogue box pops up.
4. Go to the line with your email and hit `m`. Navigate to the `[OK]` button and hit enter. Yes, the user experience for this part is abominable.
5. Kill the buffer. Open the file again.
6. Enter the password that you used when setting up your GnuPG key.
7. You can now save, close, and reopen the file multiple times without having to enter your password. You can edit other GPG files completely transparently as well, as long as your password cache doesn't expire.

If something didn't go as expected, make sure you followed this guide down to the button. If you deviated at all, I can't really help you, because I have no idea what I'm doing, and I'm not your personal tech support guy, anyway.

Conclusion

Ah, another fantastic guide.

Footnotes

[1]: Or vanilla, if you prefer. ↩

[2]: This is how I manage all of my important dot-files, actually. I plan on writing a short blog post about that. ↩

Related posts:
› Secure GPG In Emacs, Featuring Agent Smith

Filed under emacs cryptography

top ⇡

⇠ Managing Your Life With Org-Mode

Deep Work ⇢

Add a comment

M↓ MARKDOWN

COMMENT ANONYMOUSLY

ADD COMMENT

Upvotes **Newest** Oldest

Eugeny Rozhkov

0 points · 15 months ago

> And something about a “key recipient dialogue”? I have no clue.

This setting is requred for **epa** to encrypt your file with the default recipient (the next option defines that recipient and, by extension, their public key) and not show the key selection dialogue. I found out that it works as intended if you set it to any other value than *'t'* or *'nil'*:

```
(require 'epa-file)
(setq epa-file-select-keys 1)
(setq epa-file-encrypt-to '("<YOUR EMAIL HERE>"))
```

?

Anonymous

0 points · 18 months ago

“As time goes on, I become less and less a fan of open source software and the largely talent-less developers behind it.”

This is one of those statements you’ll reflect on in the future and wish you hadn’t said. Nobody deserves such generalizations.

Powered by **Commento**