GnuPG

From ArchWiki

According to the official website (http://www.gnupg.org):

> GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows to encrypt and sign your data and communication, features a versatile key management system as well as access modules for all kinds of public key directories. GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications. A wealth of frontend applications and libraries are available. Version 2 of GnuPG also provides support for S/MIME and Secure Shell (ssh).

Related articles

Disk encryption

List of applications/Security#Encr signing, steganography

Contents

Installation

Install the `gnupg (https://www.archlinux.org/packages/?name=gnupg)` package.

This will also install `pinentry (https://www.archlinux.org/packages/?name=pinentry)`, a collection of simple PIN or passphrase entry dialogs which GnuPG uses for passphrase entry. *pinentry* is determined by the symbolic link `/usr/bin/pinentry`, which by default points to `/usr/bin/pinentry-gtk-2`.

If you want to use a graphical frontend or program that integrates with GnuPG, see List of applications/Security#Encryption, signing, steganography.

Environment variables

GNUPGHOME

`$GNUPGHOME` is used by GnuPG to point to the directory where all configuration files are stored. By default `$GNUPGHOME` is not set and your `$HOME` is used instead, thus you will find a `~/.gnupg` directory right after the install. You may change this default by setting `GNUPGHOME` it in one of your regular startup files.

Note: By default, the gnupg directory has its Permissions set to *700* and the files it contains have their permissions set to *600*. Only the owner of the directory has permission to read, write and access the files (*r*,*w*,*x*). This is for security purposes and should not be changed. In case this directory or any file inside it does not follow this security measure, you will get warnings about unsafe file and home directory permissions.

Configuration file

Default is `~/.gnupg/gpg.conf` and `~/.gnupg/dirmngr.conf`. If you want to change the default location, either run gpg this way `$ gpg --homedir path/to/file` or use `$GNUPGHOME` variable. Append in this file any long options you want. Do not write the two dashes, but simply the name of the option and required arguments. You will find a skeleton files in `/usr/share/gnupg`. These files are copied to `~/.gnupg` the first time gpg is run for any operation if it doesn't exist. Other examples are found in #See_also.

Usage

Note:

- Whenever a `<user-id>` is required in a command, it can be specified with your key ID, fingerprint, a part of your name or email address, etc. GnuPG is flexible on this.
- Some of these steps may be provided by an external program depending on your usage, such as an email client. See also List of applications/Security#Encryption, signing, steganography.

Create key pair

Generate a key pair by typing in a terminal:

```
$ gpg --full-gen-key
```

Tip: You can use `--expert` for getting alternative ciphers available (like ECC Elliptic Curve).

You will be asked several questions. In general, most users will want both a RSA (sign only) and a RSA (encrypt only) key. A keysize of 2048 is sufficient. Using 4096 "gives us almost nothing, while costing us quite a lot." (https://www.gnupg.org/faq/gnupg-faq.html#no_default_of_rsa4096)

While having an expiration date for subkeys is not technically necessary, it is considered good practice. A period of a year is generally good enough for the average user. This way even if you lose access to your keyring, it will allow others to know that it is no longer valid. Note that you can extend the expiry date after key creation without having to re-issue a new key.

Be sure to choose a secure passphrase.

Backup your private key

To backup your private key do the following:

```
$ gpg --export-secret-keys --armor <user-id> > privkey.asc
```

Place the private key in a safe place, such as a locked container or encrypted drive.

Warning: Anyone who gains access to the above exported file will be able to encrypt and sign documents as if they were you *without* needing to know your passphrase.

Export your public key

In order for others to send encrypted messages to you, they need your public key.

To generate an ASCII version of your public key (e.g. to distribute it by e-mail):

```
$ gpg --armor --output public.key --export <user-id>
```

Alternatively, or in addition, you can share your key on a keyserver.

Tip: Add `--no-emit-version` to avoid printing the version number, or add the corresponding setting to your configuration file.

Import a key

In order to encrypt messages to others, you need their public key. To import a public key to your public key ring:

```
$ gpg --import public.key
```

Alternatively, find a public key on a keyserver.

Use a keyserver

You can register your key with a public PGP key server, so that others can retrieve your key without having to contact you directly:

```
$ gpg --send-keys <key-id>
```

To import a key from a key server:

```
$ gpg --recv-keys <key-id>
```

Warning: Anyone can send keys to a keyserver, so you should not trust that the key you download actually belongs to the individual listed. You should verify the authenticity of the retrieved public key by comparing its fingerprint with one that the owner published on an independent source, such as their own blog or website, or contacting them by email, over the phone or in person. Using multiple authentication sources will increase the level of trust you can give to the downloaded key. See Wikipedia:Public key fingerprint for more information.

Tip:

- An alternative key server is `pool.sks-keyservers.net` and can be specified with `--keyserver`; see also wikipedia:Key server (cryptographic)#Keyserver examples.
- You can connect to the keyserver over Tor using `--use-tor`. `hkp://jirk5u4osbsr34t5.onion` is the onion address for the sks-keyservers pool. See this GnuPG blog post (https://gnupg.org/blog/20151224-gnupg-in-november-and-december.html) for more information.

Encrypt and decrypt

When encrypting or decrypting it is possible to have more than one private key in use. If this occurs you need to select the active key. This can be done by using the option `-u <user-id>` or by using the option `--local-user <user-id>`. This causes the default key to use to be replaced by wanted key.

To encrypt a file using ASCII armor (suitable for copying and pasting a message in text format), use:

```
$ gpg --encrypt --armor secret.txt
```

If you want to just encrypt a file, exclude `--armor`.

Tip:

- If you want to change recipient this can be done by the option `-r <user-id>` (or

`--recipient <user-id>`).

- Add `-R <user-id>` or `--hidden-recipient <user-id>` instead of `--recipient` to not put the recipient key IDs in the encrypted message. This helps to hide the receivers of the message and is a limited countermeasure against traffic analysis.
- Add `--no-emit-version` to avoid printing the version number, or add the corresponding setting to your configuration file.

Note: You can use gnupg to encrypt your sensitive documents, but only individual files at a time. If you want to encrypt directories or a whole file-system you should consider using TrueCrypt or EncFS, though you can always tarball various files and then encrypt them.

To decrypt a file, use:

```
$ gpg --decrypt secret.txt.asc
```

You will be prompted to enter your passphrase. You will need to have already imported the sender's public key to decrypt a file or message from them.

Key maintenance

Edit your key

Running the `gpg --edit-key <user-id>` command will present a menu which enables you to do most of your key management related tasks.

Some useful commands in the edit key sub menu:

```
> passwd       # change the passphrase
> clean        # compact any user ID that is no longer usable (e.g revoked or expired)
> revkey       # revoke a key
> addkey       # add a subkey to this key
> expire       # change the key expiration time
```

Type `help` in the edit key sub menu for more commands.

Tip: If you have multiple email accounts you can add each one of them as an identity, using `adduid` command. You can then set your favourite one as `primary` .

Exporting subkey

If you plan to use the same key across multiple devices, you may want to strip out your master key and only keep the bare minimum encryption subkey on less secure systems.

First, find out which subkey you want to export.

```
$ gpg -K
```

Select only that subkey to export.

```
$ gpg -a --export-secret-subkeys [subkey id]! > /tmp/subkey.gpg
```

Warning: If you forget to add the !, all of your subkeys will be exported.

At this point you could stop, but it is most likely a good idea to change the passphrase as well. Import the key into a temporary folder.

```
$ gpg --homedir /tmp/gpg --import /tmp/subkey.gpg
$ gpg --homedir /tmp/gpg --edit-key <user-id>
> passwd
> save
$ gpg --homedir /tmp/gpg -a --export-secret-subkeys [subkey id]! > /tmp/subkey.altpass.gpg
```

Note: You will get a warning that the master key was not available and the password was not changed, but that can safely be ignored as the subkey password was.

At this point, you can now use `/tmp/subkey.altpass.gpg` on your other devices.

Rotating subkeys

Warning: Never delete your expired or revoked subkeys unless you have a good reason. Doing so will cause you to lose the ability to decrypt files encrypted with the old subkey. Please **only** delete expired or revoked keys from other users to clean your keyring.

If you have set your subkeys to expire after a set time, you can create new ones. Do this a few weeks in advance to allow others to update their keyring.

Note: You do not need to create a new key simply because it is expired. You can extend the expiration date.

Create new subkey (repeat for both signing and encrypting key)

```
$ gpg --edit-key <user-id>
> addkey
```

And answer the following questions it asks (see previous section for suggested settings).

Save changes

```
> save
```

Update it to a keyserver.

```
$ gpg --keyserver pgp.mit.edu --send-keys <user-id>
```

Note: Revoking expired subkeys is unnecessary and arguably bad form. If you are constantly revoking keys, it may cause others to lack confidence in you.

List keys

To list keys in your public key ring:

```
$ gpg --list-keys
```

To list keys in your secret key ring:

```
$ gpg --list-secret-keys
```

gpg-agent

gpg-agent is mostly used as daemon to request and cache the password for the keychain. This is useful if GnuPG is used from an external program like a mail client. It can be activated by adding following line in `gpg.conf`:

```
~/.gnupg/gpg.conf
```

```
use-agent
```

This tells GnuPG to use the agent whenever it needs the password. However, the agent needs to be already running. To autostart it, add the following entry to your `.xinitrc` or `.bash_profile`. Remember to change the envfile path if you changed your `$GNUPGHOME`.

```
~/.bash_profile
```

```
envfile="$HOME/.gnupg/gpg-agent.env"
if [[ -e "$envfile" ]] && kill -0 $(grep GPG_AGENT_INFO "$envfile" | cut -d: -f 2) 2>/dev/null; then
    eval "$(cat "$envfile")"
else
    eval "$(gpg-agent --daemon --enable-ssh-support --write-env-file "$envfile")"
fi
export GPG_AGENT_INFO  # the env file does not contain the export statement
export SSH_AUTH_SOCK   # enable gpg-agent for ssh
```

Log out of the session and log back in. Check if *gpg-agent* is activated:

```
$ pgrep gpg-agent
```

Configuration

gpg-agent can be configured via `~/.gnupg/gpg-agent.conf` file. The configuration options are listed in `man gpg-agent`. For example you can change cache ttl for unused keys:

```
~/.gnupg/gpg-agent.conf
```

```
default-cache-ttl 3600
```

Tip: To cache your passphrase for the whole session, please run the following command:

```
$ /usr/lib/gnupg/gpg-preset-passphrase --preset XXXXXX
```

where XXXX is the keygrip. You can get its value when running `gpg --with-keygrip -K`. Passphrase will be stored until `gpg-agent` is restarted. If you set up `default-cache-ttl` value, it will take precedence.

Reload the agent

After changing the configuration, reload the agent by piping the `RELOADAGENT` string to `gpg-connect-agent`.

```
$ echo RELOADAGENT | gpg-connect-agent
```

The shell should print `OK`.

pinentry

Finally, the agent needs to know how to ask the user for the password. This can be set in the gpg-agent configuration file.

The default uses a gtk dialog. There are other options - see `info pinentry`. To change the dialog implementation set `pinentry-program` configuration option:

```
~/.gnupg/gpg-agent.conf
```

```

# PIN entry program
# pinentry-program /usr/bin/pinentry-curses
# pinentry-program /usr/bin/pinentry-qt
# pinentry-program /usr/bin/pinentry-kwallet

pinentry-program /usr/bin/pinentry-gtk-2
```

Tip: For using `/usr/bin/pinentry-kwallet` you have to install the `kwalletcli (https://www.archlinux.org/packages/?name=kwalletcli)` package.

After making this change, reload the gpg-agent.

Start gpg-agent with systemd user

It is possible to use the Systemd/User facilities to start the agent.

Create a systemd unit file:

```
~/.config/systemd/user/gpg-agent.service
```

```
[Unit]
Description=GnuPG private key agent
IgnoreOnIsolate=true

[Service]
Type=forking
ExecStart=/usr/bin/gpg-agent --daemon --homedir=%h/.gnupg
ExecStop=/usr/bin/pkill gpg-agent
Restart=on-abort

[Install]
WantedBy=default.target
```

Note:

- You may need to set some environment variables for the service, for example `GNUPGHOME`. See systemd/User#Environment variables for details.
- if your gnupg home directory is ~/.gnupg, there is no need to specify its path
- `gpg -agent` will not use standard socket, but rather listen for a socket name `S.gpg-agent` located in your gnupg home directory. We can thus forget any script to read an environment file and get the path of the random socket created in `/tmp`.

- If you use SSH capabilities of gpg-agent (--enable-ssh-support), the systemd unit above will not work

Tip:

To ensure your gpg-agent is running and listening to connection, simply run this command: `$ gpg-connect-agent` . If your settings are valid, you will be on a prompt (enter *bye* and *quit* to close connection and leave)

Unattended passphrase

Starting with GnuPG 2.1.0 the use of gpg-agent and pinentry is required; this may break backwards compatibility for passphrases piped in from STDIN using the `--passphrase-fd 0` commandline option. In order to have the same type of functionality as the older releases two things must be done:

First, edit the gpg-agent configuration to allow *loopback* pinentry mode:

```
~/.gnupg/gpg-agent.conf
```

```
allow-loopback-pinentry
```

Restart the gpg-agent process if it is running to let the change take effect.

Second, either the application needs to be updated to include a commandline parameter to use loopback mode like so:

```
$ gpg --pinentry-mode loopback ...
```

...or if this is not possible, add the option to the configuration:

```
~/.gnupg/gpg.conf
```

```
pinentry-mode loopback
```

Note: The upstream author indicates setting **pinentry-mode loopback** in *gpg.conf* may break other usage, using the commandline option should be preferred if at all possible. [1] (https://bugs.g10code.com/gnupg/issue1772)

Smartcards

Note: `pcsclite (https://www.archlinux.org/packages/?name=pcsclite)` and `libusb-compat (https://www.archlinux.org/packages/?name=libusb-compat)` have to be installed, and the contained systemd service `pcscd.service` has to be running.

GnuPG uses *scdaemon* as an interface to your smartcard reader, please refer to the man page for details.

GnuPG only setups

If you do not plan to use other cards but those based on GnuPG, you should check the `reader-port` parameter in `~/.gnupg/scdaemon.conf` . The value '0' refers to the first available serial port reader and a value of '32768' (default) refers to the first USB reader.

GnuPG together with OpenSC

If you are using any smartcard with an opensc driver (e.g.: ID cards from some countries) you should pay some attention to GnuPG configuration. Out of the box you might receive a message like this when using `gpg --card-status`

```
gpg: selecting openpgp failed: ec=6.108
```

By default, scdaemon will try to connect directly to the device. This connection will fail if the reader is being used by another process. For example: the pcscd daemon used by OpenSC. To cope with this situation we should use the same underlying driver as opensc so they can work well together. In order to point scdaemon to use pcscd you should remove `reader-port` from `~/.gnupg/scdaemon.conf`, specify the location to `libpcsclite.so` library and disable ccid so we make sure that we use pcscd:

```
~/.gnupg/scdaemon.conf
```

```
pcsc-driver /usr/lib/libpcsclite.so
card-timeout 5
disable-ccid
```

Please check `man scdaemon` if you do not use OpenSC.

Tips and tricks

Different algorithm

You may want to use stronger algorithms:

```
~/.gnupg/gpg.conf
```

```
...

personal-digest-preferences SHA512
cert-digest-algo SHA512
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP Uncompressed
personal-cipher-preferences TWOFISH CAMELLIA256 AES 3DES
```

In the latest version of GnuPG, the default algorithms used are SHA256 and AES, both of which are secure enough for most people. However, if you are using a version of GnuPG older than 2.1, or if you want an even higher level of security, then you should follow the above step.

Encrypt a password

It can be useful to encrypt some password, so it will not be written in clear on a configuration file. A good example is your email password.

First create a file with your password. You **need** to leave **one** empty line after the password, otherwise gpg will return an error message when evaluating the file.

Then run:

```
$ gpg -e -a -r <user-id> your_password_file
```

`-e` is for encrypt, `-a` for armor (ASCII output), `-r` for recipient user ID.

You will be left with a new `your_password_file.asc` file.

Default options for new users

If you want to setup some default options for new users, put configuration files in `/etc/skel/.gnupg/`. When the new user is added in system, files from here will be copied to its GnuPG home directory. There is also a simple script called *addgnupghome* which you can use to create new GnuPG home directories for existing users:

```
# addgnupghome user1 user2
```

This will add the respective `/home/user1/.gnupg` and `/home/user2/.gnupg` and copy the files from the skeleton directory to it. Users with existing GnuPG home directory are simply skipped.

Revoking a key

Warning:

- Anybody having access to your revocation certificate can revoke your key, rendering it useless.
- Key revocation should only be performed if your key is compromised or lost, or you forget your passphrase.

Revocation certificates are automatically generated for newly generated keys, although one can be generated manually by the user later. These are located at `~/.gnupg/openpgp-revocs.d/`. The filename of the certificate is the fingerprint of the key it will revoke.

To revoke your key, simply import the revocation certificate:

```
$ gpg --import <fingerprint>.rev
```

Now update the keyserver:

```
$ gpg --keyserver subkeys.pgp.net --send <userid>
```

Change trust model

By default GnuPG uses the Web of Trust as the trust model. You can change this to Trust on First Use by adding `--trust-model=tofu` when adding a key or adding this option to your GnuPG configuration file. More details are in this email to the GnuPG list (https://lists.gnupg.org/pipermail/gnupg-devel/2015-October /030341.html).

Hide all recipient id's

By default the recipient's key ID is in the encrypted message. This can be removed at encryption time for a recipient by using `hidden-recipient <user-id>`. To remove it for all recipients add `throw-keyids` to your configuration file. This helps to hide the receivers of the message and is a limited countermeasure against traffic analysis. (Using a little social engineering anyone who is able to decrypt the message can check whether one of the other recipients is the one he suspects.) On the receiving side, it may slow down the decryption process because all available secret keys must be tried (*e.g.* with `--try-secret-key <user-id>`).

Using caff for keysigning parties

To allow users to validate keys on the keyservers and in their keyrings (i.e. make sure they are from whom they claim to be), PGP/GPG uses he Web of Trust. Keysigning parties allow users to get together in physical location to validate keys. The Zimmermann-Sassaman key-signing protocol is a way of making these very effective. Here (http://www.cryptnet.net/fdp/crypto/keysigning_party/en/keysigning_party.html) you will find a how-to article.

For an easier process of signing keys and sending signatures to the owners after a keysigning party, you can use the tool *caff*. It can be installed from the AUR with the package `caff-svn (https://aur.archlinux.org/packages/caff-svn/)` AUR or bundled together with other useful tools in the package `signing-party-svn (https://aur.archlinux.org/packages/signing-party-svn/)` AUR[broken link: archived in aur-mirror (http://pkgbuild.com/git/aur-mirror.git/tree/signing-party-svn)]. Either way, there will be a lot of dependencies installing from the AUR. Alternatively you can install them from CPAN with

```
cpanm Any::Moose
cpanm GnuPG::Interface
```

To send the signatures to their owners you need a working MTA. If you do not have already one, install msmtp.

Troubleshooting

Make it work behind an http proxy

Since 2.1.9 the http proxy option can be set like this:

```
 gpg --keyserver-option http-proxy=HOST:PORT
```

See https://bugs.gnupg.org/gnupg/issue1786 for more explanation.

Not enough random bytes available

When generating a key, gpg can run into this error:

```
Not enough random bytes available. Please do some other work to give the OS a chance to collect more entropy!
```

To check the available entropy, check the kernel parameters:

```
cat /proc/sys/kernel/random/entropy_avail
```

A healthy Linux system with a lot of entropy available will have return close to the full 4,096 bits of entropy. If the value returned is less than 200, the system is running low on entropy.

To solve it, remember you do not often need to create keys and best just do what the message suggests (e.g. create disk activity, move the mouse, edit the wiki - all will create entropy). If that does not help, check which service is using up the entropy and consider stopping it for the time. If that is no alternative, see Random number generation#Faster alternatives.

su

When using `pinentry`, you must have the proper permissions of the terminal device (e.g. `/dev/tty1`) in use. However, with *su* (or *sudo*), the ownership stays with the original user, not the new one. This means that pinentry will fail, even as root. The fix is to change the permissions of the device at some point before the use of pinentry (i.e. using gpg with an agent). If doing gpg as root, simply change the ownership to root right before using gpg:

```
chown root /dev/ttyN  # where N is the current tty
```

and then change it back after using gpg the first time. The equivalent is likely to be true with `/dev/pts/`.

Note: The owner of tty *must* match with the user for which pinentry is running. Being part of the group `tty` **is not** enough.

Agent complains end of file

The default pinentry program is pinentry-gtk-2, which needs a DBus session bus to run properly. See General troubleshooting#Session permissions for details.

Alternatively, you can use `pinentry-qt`. See #pinentry.

KGpg configuration permissions

There have been issues with `kdeutils-kgpg (https://www.archlinux.org/packages/?name=kdeutils-kgpg)` being able to access the `~/.gnupg/` options. One issue might be a result of a deprecated *options* file, see the bug (https://bugs.kde.org/show_bug.cgi?id=290221) report.

Another user reported that *KGpg* failed to start until the `~/.gnupg` folder is set to `drwxr-xr-x` permissions. If you require this work-around, ensure that the directory contents retain `-rw-------` permissions! Further, report it as a bug to the developers (https://bugs.kde.org/buglist.cgi?quicksearch=kgpg).

Conflicts between gnome-keyring and gpg-agent

While the Gnome keyring implements a GPG agent component, as of GnuPG version 2.1, GnuPG ignores the `GPG_AGENT_INFO` environment variable, so that Gnome keyring can no longer be used as a GPG agent.

However, since version 0.9.6 the package `pinentry (https://www.archlinux.org/packages/?name=pinentry)` provides the `pinentry-gnome3` program. You may set the following option in your `gpg-agent.conf` file

```
 pinentry-program /usr/bin/pinentry-gnome3
```

in order to make use of that pinentry program.

Since version 0.9.2 all pinentry programs can be configured to optionally save a passphrase with libsecret. For example, when the user is asked for a passphrase via `pinentry-gnome3`, a checkbox is shown whether to save the passphrase using a password manager. Unfortunately, the package `pinentry (https://www.archlinux.org/packages/?name=pinentry)` does not have this feature enabled (see FS#46059 (https://bugs.archlinux.org/task/46059) for the reasons). You may use `pinentry-libsecret (https://aur.archlinux.org/packages/pinentry-libsecret/)`^{AUR} as a replacement for it, which has support for libsecret enabled.

mutt and gpg

To be asked for your GnuPG password only once per session as of GnuPG 2.1, see this forum thread (https://bbs.archlinux.org/viewtopic.php?pid=1490821#p1490821).

"Lost" keys, upgrading to gnupg version 2.1

When `gpg --list-keys` fails to show keys that used to be there, and applications complain about missing or invalid keys, some keys may not have been migrated to the new format.

Please read GnuPG invalid packet workaround (http://jo-ke.name/wp/?p=111). Basically, it says that there is a bug with keys in the old `pubring.gpg` and `secring.gpg` files, which have now been superseded by the new `pubring.kbx` file and the `private-keys-v1.d/` subdirectory and files. Your missing keys can be recovered with the following commnads:

```
$ cd
$ cp -r .gnupg gnupgOLD
$ gpg --export-ownertrust > otrust.txt
$ gpg --import .gnupg/pubring.gpg
$ gpg --import-ownertrust otrust.txt
$ gpg --list-keys
```

gpg hanged for all keyservers (when trying to receive keys)

If gpg hanged with a certain keyserver when trying to receive keys, you might need to kill dirmngr in order to get access to other keyservers which are actually working, otherwise it might keeping hanging for all of them.

Smartcard not detected

Your user might not have the permission to access the smartcard which results in a `card error` to be thrown, even though the card is correctly set up and inserted.

One possible solution is to add a new group `scard` including the users who need access to the smartcard.

Then use an udev rule, similar to the following:

```
/etc/udev/rules.d/71-gnupg-ccid.rules
```

```
ACTION=="add", SUBSYSTEM=="usb", ENV{ID_VENDOR_ID}=="1050", ENV{ID_MODEL_ID}=="0116|0111", MODE="664", GROUP="scard"
```

One needs to adapt VENDOR and MODEL according to the `lsusb` output, the above example is for a YubikeyNEO.

See also

- GNU Privacy Guard Homepage (https://gnupg.org/)
- Creating GPG Keys (Fedora) (https://fedoraproject.org/wiki/Creating_GPG_Keys)
- OpenPGP subkeys in Debian (https://wiki.debian.org/Subkeys)
- A more comprehensive gpg Tutorial (http://blog.sanctum.geek.nz/series/linux-crypto/)
- gpg.conf recommendations and best practices (https://help.riseup.net/en/security/message-security/openpgp/gpg-best-practices)
- Torbirdy gpg.conf (https://github.com/ioerror/torbirdy/blob/master/gpg.conf)
- /r/GPGpractice - a subreddit to practice using GnuPG. (https://www.reddit.com/r/GPGpractice/)