Setting up a GPG Identity

Your GPG key is your "ID Card" of the future, that *you* control. This is how to set up and secure it for use with your laptop, desktop, and mobile.

T**he internet**, as a concept and transport layer, does not need to know about human identity. It only needs to know about the address and authenticity of other computers, if it needs to connect to them. It works better this way. Still, on and off the internet, it sometimes becomes necessary to address the concern of human identity. GPG should be explored for that concern. GPG stands for The *GNU Privacy Guard*, or GnuPG. You may have heard of PGP, a modest acronym for "Pretty Good Privacy," and GPG is essentially a more robust implementation of the OpenPGP protocol and is similar to PGP. GPG deserves attention for facilitating identity. Additionally, GPG provides a very secure encryption protocol. Your GPG key will already have widespread email client support for encrypting messages, and it can be used for general file encryption on your device.

You can create and manage multiple GPG keys. For example, you might want to create a separate GPG key for your personal email address and your work email address for use with respective applications. If you need to, you can associate multiple email addresses with one GPG identity.

After you complete the outlined steps via the CLI, you can manage your GPG keys with an app such as Seahorse or another GPG GUI frontend.

At the time of this writing, the GPG client version is `2.2.4`.

The process outlined below is tested so far only on Linux. E.g. Mint, Ubuntu, Debian. GPG is cross-platform. The author would imagine, but not guarantee, that the same CLI steps will also work on both Windows and Mac OSX.

Enter the GPG ID you will be using: user@example.org

This is only used in this article, and it is for making the commands easier to copy and paste. That is, except when you need to specify particular subkey IDs. Your email is not stored or submitted anywhere.

If you do not enter a GPG ID or email in the input above, this guide will use `user@example.org`.

Let's get started on creating a new GPG key:

```
$ gpg --expert --full-generate-key
```

The `--full-generate-key` option forces GPG to provide you with all the default wizard configuration options for creating a new key. The `--expert` option gives us a few more selections we'll use. We want to use the strongest cypher length available, which is 4096 at the time of this writing. Using only `--gen-key` does not always provide all of the options.

After running the above command, complete the following steps:

```
> (8) RSA (set your own capabilities)
> Select: (S) Toggle the sign capability
> Select: (E) Toggle the encrypt capability
> Select: (Q) Finished
> Keysize: 4096
> Key expiry: 5y
> Confirm
> Enter Name and Email, optional comment
> Okay
> Create/enter passphrase
> Generate entropy
```

This gives us a keypair that is able to certify other keys. We can use this ability to create new subkeys. We'll create one encryption subkey that all devices will use, and a separate signing key for each device. We are going to create the subkeys for each device from the device we're at now, assuming the current device is the most "trusted," e.g. a desktop that stays at home.

Next we'll edit the newly-created key, for the purpose of updating it to prefer stronger ciphers.

```
$ gpg --expert --edit-key user@example.org
```

When GPG has loaded its own command line interface to edit the specified key, run the following command:

```
gpg> setpref SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES
```

```
> Confirm
> Enter passphrase
```

```
gpg> save
```

Add the encryption subkey. This will be used by all the devices.

In some cases you don't need to encrypt from other devices, only sign, but in this example we are creating and exporting an encryption key that all the devices will use.

```
$ gpg --expert --edit-key user@example.org
```

```
gpg> addkey
```

```
> (8) RSA (set your own capabilities)
> Select: (S) Toggle the sign capability
> Select: (Q) Finished
> Keysize: 4096
> Key expiry: 1y
> Confirm
> Confirm
> Enter passphrase
> Generate entropy
```

Add the signing subkeys for each device.

```
$ gpg --expert --edit-key user@example.org
```

```
gpg> addkey
```

```
> (8) RSA (set your own capabilities)
> (E) Toggle the encrypt capability
> Select: (Q) Finished
> Keysize: 4096
> Key expiry: 1y
> Confirm
> Confirm
> Enter passphrase
> Generate entropy
```

Repeat this `addkey` process for each device that needs a subkey right now. When we strip the master key secrets from the GPG key later, we won't be able to add new subkeys with `addkey` unless we re-import our master key secret. That is why we are adding them all now for all the devices that we are setting up now.

For day-to-day use on individual devices, we can export only the subkeys from the GPG key, without exporting the whole master key. This way our master key secret data will not be unnecessarily transferred to other devices.

Create a revocation certificate. This is in case of emergency, and it is highly recommeded that you create and backup a revocation certificate:

```
$ gpg --output user-example.org.master.gpg-revocation-cert
```

```
> Confirm
> Reason: 0 = No reason specified
> Optional description
> Confirm
```

We will now export the public and private keys and subkeys for the master GPG keypair for backup. Our master key is currently created and stored from the commands outlined above. We might need it again in the future for creating new subkeys after we finish creating our subkeys today, or to sign other people's keys. You should keep a backup of your original master keypair. The original master keypair should be treated as very sensitive data.

When we are creating and transferring subkeys for other devices, we will be stripping the master keypair's *secret* data. But first we want a backup of our original master key, to be kept safe.

Export the master private key:

```
$ gpg --export-secret-keys --armor user@example.org > user
```

```
> Enter passphrase
```

Export all the private subkeys:

```
$ gpg --export-secret-subkeys --armor user@example.org > u
```

Export the master/main public key and public subkeys:

```
$ gpg --export --armor user@example.org > user-example.org
```

Master keypair files are now saved/exported. Back it up as securely as you can.

We will now strip the master key secrets. We should only need to use that in the future if we are creating new keys, revoking keys, as well as some other tasks that only the master key can authorize.

There aren't currently a lot of more straightforward ways to just strip the master secrets, so we will delete ALL secret keys from the ring, and then import back only our subkey secrets that we saved.

```
$ gpg --delete-secret-keys user@example.org
```

```
> Confirm (multiple times)
```

```
$ gpg --import user-example.org.subkeys.private.gpg-key
```

Now we can export the (common) private encryption and (separate) private signing subkey for each device.

First we'll need to get a list of the subkey IDs.

```
$ gpg --expert --edit-key user@example.org
```

You will see a list of all the subkeys associated with this GPG ID:

```
pub  rsa4096/5E6400304CF5249E
     created: 2018-09-01  expires: 2023-08-31  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa4096/AC046286CB3DA99A
     created: 2018-09-01  expires: 2019-09-01  usage: E
ssb  rsa4096/CE2347C133F78AA3
     created: 2018-09-01  expires: 2019-09-01  usage: S
ssb  rsa4096/8DB3CD6F9A3E3E34
     created: 2018-09-01  expires: 2019-09-01  usage: S
ssb  rsa4096/9EB12545F1B50E90
     created: 2018-09-01  expires: 2019-09-01  usage: S
```

Now quit GPG because all we needed is the list:

```
gpg> quit
```

In this example, our common encryption key is `AC046286CB3DA99A`. And we are using the following for signing keys on the devices:

`CE2347C133F78AA3` - Desktop (current machine)
`8DB3CD6F9A3E3E34` - Laptop
`9EB12545F1B50E90` - Phone

For each device, export both the shared private encryption key and the device's private signing key.

For maximizing security, you can create a single-use password to protect your exported file during the transfer/import process. When you import your key into your other device, it will ask for the single use/one-time password.

Here is the example for the laptop encryption and signing keys:

```
$ gpg --export-secret-subkeys --armor AC046286CB3DA99A! 8D
```

```
> Enter your new SINGLE-USE PASSWORD
> Enter passphrase (for GPG key)
> Confirm your SINGLE-USE PASSWORD
```

Note the exclamation points (`!`) at the end of the IDs, denoting that they are specific subkey IDs and not the master key ID.

If you do not need to use a one-time password to secure the transfer of your export, you can use this command instead:

```
$ gpg --export-secret-subkeys --armor AC046286CB3DA99A! 8D
```

Repeat for the mobile phone:

```
$ gpg --export-secret-subkeys --armor AC046286CB3DA99A! 9E
```

Or the mobile phone without the one-time password:

```
$ gpg --export-secret-subkeys --armor AC046286CB3DA99A! 9E
```

IMPORTING SUBKEY EXPORT FILE

This will import *just* the subkeys that we exported previously. For day-to-day use, that is usually all we need for our devices.

If you have protected your export file with a one-time passphrase, use this method. For the laptop example:

```
$ gpg --decrypt user-example.org.subkeys-laptop.private.gp
```

```
> Enter one-time decryption passphrase
> Enter GPG key passphrase
```

If the export file is not encrypted, this is the command to import:

```
$ gpg --import user-example.org.subkeys-laptop.private.gpg
```

```
> Enter GPG key passphrase
```

For the phone example, the only difference in the commands is the filename:

```
$ gpg --decrypt user-example.org.subkeys-phone.private.gpg
```

And without a one-time passphrase:

```
$ gpg --import user-example.org.subkeys-phone.private.gpg
```

SETTING THE TRUST LEVEL OF IMPORTED KEYS

Our original created master key was trusted by default, but after importing keys we will need to manually set the trust level for them.

This is required for some applications that utilize GPG and should generally be performed.

```
$ gpg --expert --edit-key user@example.org
```

```
gpg> trust
```

```
> Select: 5 = I trust ultimately
```

```
gpg> save
```

Delete the signing subkeys we added that are not intended for the current device

You can use a GUI program such as Seahorse (Linux) or OpenKeychain (Android) for this deletion process as well. We just want to strip any subkeys that are going to be used for other devices, keeping only the singing subkey we will use for the current device.

```
$ gpg --expert --edit-key user@example.org
```

You will see a list of keys associated with the ID `user@example.org`. In the following example, **3 new signing subkeys were added**, noted with `usage: S`. The one marked with `usage: C` is the certify one we first created, and the one marked with `usage: E` is the encryption key we created. You might see one marked with `usage: SC` (sign and certify) if you did not use this guide to create the initial key.

```
pub  rsa4096/5E6400304CF5249E
     created: 2018-09-01  expires: 2023-08-31  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa4096/AC046286CB3DA99A
     created: 2018-09-01  expires: 2019-09-01  usage: E
ssb  rsa4096/CE2347C133F78AA3
     created: 2018-09-01  expires: 2019-09-01  usage: S
ssb  rsa4096/8DB3CD6F9A3E3E34
     created: 2018-09-01  expires: 2019-09-01  usage: S
ssb  rsa4096/9EB12545F1B50E90
     created: 2018-09-01  expires: 2019-09-01  usage: S
```

Select the signing keys to delete using the `key` command.

```
gpg> key [N]
```

`[N]` is the zero-index of the key you wish to delete. It will be greater than zero since the first entry is your master public key and cannot be selected.

For this example, we are using the following devices:

`CE2347C133F78AA3` - Desktop (current machine)
`8DB3CD6F9A3E3E34` - Laptop
`9EB12545F1B50E90` - Phone

We will issue the following command to select the laptop and mobile phone subkeys that will be deleted from the desktop:

```
gpg> key 3
gpg> key 4
```

Notice now in the list, there is an asterisk `*` indicating which subkeys are selected:

```
pub  rsa4096/5E6400304CF5249E
     created: 2018-09-01  expires: 2023-08-31  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa4096/AC046286CB3DA99A
     created: 2018-09-01  expires: 2019-09-01  usage: E
ssb  rsa4096/CE2347C133F78AA3
     created: 2018-09-01  expires: 2019-09-01  usage: S
ssb* rsa4096/8DB3CD6F9A3E3E34
     created: 2018-09-01  expires: 2019-09-01  usage: S
ssb* rsa4096/9EB12545F1B50E90
     created: 2018-09-01  expires: 2019-09-01  usage: S
```

Once the correct subkeys are selected, issue the `delkey` command to delete them, and then `save`.

```
gpg> delkey
gpg> save
```

Now you should be all setup on your devices, and your master key is preserved in backup.

References:

https://gnupg.org/documentation/manpage.html

https://alexcabal.com/creating-the-perfect-gpg-keypair/

https://www.paulfurley.com/gpg-for-humans-protecting-your-primary-key/

https://www.openkeychain.org/faq/