m a d b o a . c o m

# GPG Quick Start

**Paul Heinlein** <*heinlein@madboa.com*>
Initial publication: July 7, 2004
Most recent revision: July 20, 2011

A quick **gpg** HOWTO for getting started with GnuPG.

**Table of Contents**

---

A colleague at work once asked me how to get started using **gpg**, the GNU Privacy Guard. He had no experience with it at all. Here's a slightly expanded version of what I told him.

## Your Key

Private and public keys are at the heart of **gpg**'s encryption and decryption processes. The best first step is to create a key pair for yourself.

1. Generate a private key.

   ```
   gpg --gen-key
   ```

   You'll have to answer a bunch of questions:

   a. What kind and size of key you want; the defaults are probably good enough.

   b. How long the key should be valid. You can safely choose a non-expiring key for your own use. If you plan to use a key for public signing, you might want to consider a yearly expiration.

   c. Your real name and e-mail address; these are necessary for identifying your key in a larger set of keys.

   d. A comment for your key, perhaps to distinquish a key used for special tasks like signing software releases. The comment can be empty.

   e. A passphrase. Whatever you do, *don't forget it!* Your key, and all your encrypted files, will be useless if you do.

2. Generate an ASCII version of your public key.

```
gpg --armor --output pubkey.txt --export 'Your Name'
```

You can freely distribute this file by sending it to friends, posting it on your web site, or whatever.

3. You might also want to register your key with public keyservers so that others can retrieve your key without having to contact you directly.

```
gpg --send-keys 'Your Name' --keyserver hkp://subkeys.pgp.net
```

## Encrypting a file for personal use

Encrypting files for your personal use is quite easy.

1. Encrypt a file called `foo.txt`. The argument to the `--recipient` option should be the all or part of the name you used when generating your private key.

```
# the long version
gpg --encrypt --recipient 'Your Name' foo.txt

# using terse options
gpg -e -r Name foo.txt
```

The encrypted version of the file will by default be named `foo.txt.gpg`. You can modify that behavior using the `--output` (`-o`) option.

2. Decrypt the encrypted file. You’ll be asked to provide the passphrase you used when generating your private key. If you don’t use the `--output` option, the contents of the encrypted file will be sent to standard output.

```
gpg --output foo.txt --decrypt foo.txt.gpg
```

If you have an encrypted file that you think you’ll want to edit on a regular basis, you might consider using the gnupg.vim plugin for transparently editing gpg-encrypted files.

A more DIY approach can use **make** to automate the process of viewing and editing your encrypted file. Here’s an example `Makefile` (to use it, you’ll need to make sure that the leading whitespace in the targets is composed of Tabs, not ordinary spaces). The example assumes that `foo.txt` is the name of the unencrypted version of your file.

```
# example Makefile for viewing/editing an encrypted file
GPGID = you@your.address
FILEPLAIN = foo.txt
FILECRYPT = $(FILEPLAIN).gpg

GPG = gpg
RM = /bin/rm -i
VI = vim

all:
	@echo ""
```

```
        @echo "usage:"
        @echo ""
        @echo "* make view -- to see $(FILEPLAIN)"
        @echo "* make edit -- to edit $(FILEPLAIN)"
        @echo ""

edit:
        @umask 0077;\
          $(GPG) --output $(FILEPLAIN) --decrypt $(FILECRYPT)
        @$(VI) $(FILEPLAIN)
        @umask 0077;\
          $(GPG) --encrypt --recipient $(GPGID) $(FILEPLAIN)
        @$(RM) $(FILEPLAIN)

view:
        @umask 0077; $(GPG) --decrypt $(FILECRYPT) | less
```

## Encrypting a file for someone else

The really cool thing about GnuPG is that you can safely encrypt files for others using publicly available keys.

1. Import your friend's key, which you might have received via e-mail or on a floppy. If the file is named `key.asc`, then just use the `--import` option to add it to your keyring:

```
gpg --import key.asc
```

   That's it! You can verify the import using the `--list-keys` option.

2. Alternatively, you might be able to find your friend's key on a public keyserver.

```
gpg --search-keys 'myfriend@his.isp.com' \
  --keyserver hkp://subkeys.pgp.net
```

   Here's what a session looks like when someone searches for my key.

```
$ gpg --search-keys heinlein@madboa
gpgkeys: WARNING: this is an *experimental* HKP interface!
gpgkeys: searching for "heinlein@madboa" from HKP server
subkeys.pgp.net
Keys 1-5 of 5 for "heinlein@madboa"
(1)     Paul Heinlein <heinlein@madboa.com>
          1024 bit DSA key 8F54CA35, created 2000-11-10
(2)     Paul Heinlein <heinlein@ohsu.edu>
          1024 bit DSA key 8F54CA35, created 2000-11-10
(3)     Paul Heinlein <heinlein@cse.ogi.edu>
          1024 bit DSA key 8F54CA35, created 2000-11-10
(4)     Paul Heinlein <heinlein@teleport.com>
          1024 bit DSA key 8F54CA35, created 2000-11-10
(5)     [user attribute packet]
          1024 bit DSA key 8F54CA35, created 2000-11-10
Enter number(s), N)ext, or Q)uit > 1
gpgkeys: WARNING: this is an *experimental* HKP interface!
gpg: key 8F54CA35: public key "Paul Heinlein <heinlein@madboa.com>"
```

```
imported
gpg: Total number processed: 1
gpg:               imported: 1
```

You'll note that my key has four different e-mail addresses attached to it. That's perfectly normal.

3. Once you've got the other person's public key, encrypt a file using it.

```
gpg --encrypt --recipient 'myfriend@his.isp.net' foo.txt
```

You'll end up with a file called `foo.txt.gpg` that you can send as an e-mail attachment or make available for downloading via ftp or the web.

## Decrypting a file from someone else

If someone sends you an encrypted file, the file has typically been encrypted using your public key. Decrypting it is no different than decrypting a file you've encrypted for your own use.

```
gpg --output foo.txt --decrypt foo.txt.gpg
```

## Detached Signatures

GnuPG can come in handy when you want to be assured that the file you've just downloaded is the one its creator wants you to have. The OpenVPN developers, for instance, release GnuPG signatures for all their downloads.

To verify a file using its detached signature, you must first have imported the signer's public key. Assume we've downloaded `crucial.tar.gz` and the developers have also released a signature file, `crucial.tar.gz.asc`. Once you're confident that you have the developers' public key in your local keyring, then the verification step is easy:

```
gpg --verify crucial.tar.gz.asc crucial.tar.gz
```

Creating a detached signature is similarly easy. The following example will create a signature for `your-file.zip` called `your-file.zip.asc`.

```
gpg --armor --detach-sign your-file.zip
```

People who have imported your public key into their keyrings can then verify that their version of your file is identical to theirs.

## Basic Key Management

After a while, you will probably have several keys in your ring. It's easy to list them all:

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

Should you lose trust in or contact with a person with a key in your ring, you'll want to delete it:

```
gpg --delete-key 'myfriend@his.isp.com'
```

## For further reading

To move beyond these simple instructions, consult the GnuPG Documentation.

return to technical writings
home - tech - praise - paul - books - about
printer-friendly layout