Howto use PGP

Pieter de Boer

October 22, 2004

1 Preface

In this document I will describe the procedure of generating a PGP key and using the key for signing and encrypting messages. By following this document, a reader should be able to set up his or her own PGP key pair, publish it on the internet and use it for verification, signing and encrypting purposes.

2 PGP software installation

There are several PGP implementations to be found. One of them is the GnuPG system, an open source, GPL licensed implementation of the OpenPGP standard. I'll be using GnuPG throughout this article.

Many systems already come shipped with GnuPG, but some do not. Based on your Operating System and — in case of Linux — distribution thereof, you may have to install GnuPG yourself. Installing GnuPG on my laptop proved to be an easy task. On Debian Linux GnuPG can be installed with:

```
apt-get install gnupg
```

To check if your installation was succesful, run:

```
gpg -h
```

If GnuPG was successfully installed, a list of options should be displayed.

3 Generating a keypair

Before you generate a keypair, gpg first needs to create the /.gnupg directory, /.gnupg/gpg.conf, /.gnupg/secring.pgp and /.gnupg/pubring.pgp. You can let gpg take care of this by simply issueing the command "gpg", followed with a ^C, to stop the tool. Gpg will create the directory and files for you.

After the directory and files are built, generating a keypair is as easy as running the following command:

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

You will be asked a few questions, which you will need to answer according to your needs. I have chosen to use the default key types `DSA` and ElGamal. The `DSA` key was 1024 bits big, I configured the `ELG-E` key to be 2048 bits big, the highest suggested keysize. If you want your key to expire after a certain amount of time, you can tell the program to do so. After filling in these options, you have to configure the visible parts of your key, being your name, an e-mail address, and if you want, a comment describing the key. If your personal information is configured, the program asks for a passphrase. The passphrase is needed to access your private key, so it is vitally important that your passphrase is a strong one. After providing the passphrase, gpg will generate the keypair.

4 Publishing the public key

To be able to use the keys for your communication, the public key needs to be published to `PGP` servers on the internet. First, you will have to edit your /.gnupg/gpg.conf file and add some servers gpg will use. I added the following lines to my configuration:

```
keyserver ldap://certserver.pgp.com
keyserver http://pgpkeys.mit.edu:11371
keyserver ldap://keyserver.pgp.com
keyserver ldap://europe.keys.pgp.com:11370
keyserver hkp://subkeys.pgp.net
```

Now you're ready to publish your key. This can be done by using the *–send-keys* option to gpg. It needs an argument: your e-mail address. I used the following command:

```
gpg --send-keys pieter@os3.nl
```

The program will give notice if anything goes wrong.

5 Signing a file

Signing a file is very easy, if you've remembered your passphrase, at least. Without your passphrase `PGP` is useless, so you have to make sure you remember it. To sign a text file with a cleartext key, use the *–clearsign* option. This option needs the file to sign as argument and will create a signed copy of the file you want to sign, with .asc appended to the filename. Let's say you want to sign the file "mail.txt". The following command could be used to do just that:

```
gpg --clearsign mail.txt
```

This command will create a cleartext signed file "mail.txt.asc", which has the original text inside, with a PGP signature added to it.

6 Verifying a signature

Verifying a signature is even easier than signing a file. Let's continue using the example above. Since you signed the file yourself, a verify should result in a good signature. To verify a signature, use the *–verify* option, using the signed file as argument. If you have a file with a so-called 'detached signature', which means that the signature is in a seperate file, then use the signature file and then the signed file as arguments to the *–verify* option. An example:

```
$ gpg --verify mail.txt.asc
gpg: Signature made Fri Oct 22 16:14:55 2004 CEST using DSA key ID B1FC47B4
gpg: Good signature from "Pieter de Boer <pieter@os3.nl>"
```

As you can see, the signature is said to be good, so this file really was signed by me.

7 Encrypting a file

Encrypting a file can be done using the *-e* option. If the *-a* option is used too, a 7 bits ascii file is created. Without that option the encrypted file will be 8 bits ascii. You will need to use 7 bits ascii to send the encrypted file using e-mail. Use the *-r* option to select the user you want to send the file to. Let's say I wanted to send myself an encrypted e-mail. I could use the following command:

```
gpg -a -r pieter@os3.nl -e mail.txt
```

A new file, called "mail.txt.asc" will be created, containing the encrypted message.

8 Decrypting a file

Decrypting can be done by using the *-d* option. It needs one argument, the file to decrypt. Continueing on the above example, to decrypt the file "mail.txt.asc", simply issue this command:

```
gpg -d mail.txt.asc
```

Again, gpg will ask for your passphrase, since it needs to unlock your private key to decrypt the message. If you have given the right passphrase, the decrypted message will be shown on stdout.

9 Editing keys

Now that we are able to sign, verify, encrypt and decrypt messages, it's time to learn about trusting keys. Trusting somebody's public key means that you have verified with that person, over a trusted medium, such as oral conversation, that indeed that key belongs to said person. The options *–edit-key* can be used to do all kinds of operations on a key, including your own. Let's say you have the key of a friend, with e-mail address john@example.com. First, edit the key you have using:

```
gpg --edit-key john@example.com
```

Now issue the command "trust". A simple menu will be shown, asking you to tell how trusted the key really is. The first option is "Don't know", the last "I trust ultimately". If you have verified that the key is correct, you can ultimately trust the key, using option 5. Next time you receive an encrypted or signed message from john@example.com and the key is correct, gpg will not only tell you that it's correct, it will also tell you that it's trusted.

One way to help build a web of trust, is by signing other people's keys with your own. This way, people who trust your key, can automatically trust the keys of people you trust. For example, if you sign john@example.com's key, anyone trusting your key can then trust his too. In the *–edit-key* screen, use the command *sign*. After signing the key, you will have to send it to the keyservers too, otherwise it won't be publicly known you trust that certain key. You can use the *–send-key* option used before to do that.

10 Conclusion

Using this howto, anyone should be able to create a `PGP` key pair using the GnuPG implementation of the OpenPGP standard, use it to sign, verify, encrypt and decrypt files and help the community by adding trust relations to the `PGP` web of trust.