Gnu Privacy Guard (GnuPG) Mini Howto (English)

Brenno J.S.A.A.F. de Winter (English) <brenno@dewinter.com> Michael Fischer v. Mollard (German) <fischer@math.uni-goettingen.de> Arjen Baart <arjen@andromeda.nl> Version 0.1.4 August 10, 2004

This documents explains how to use the GNU Privacy Guard (GnuPG), an Open Source OpenPGP compatible encryption system To keep this program totally free the use of the RSA algorithm and other patented algorithm has been avoided. The document was originally written by Michael Fischer v. Mollar in German. The text has been translated and adjusted on some points and cannot be considered as a full one-on-one copy.

Contents

1 Concepts

Remark for printed versions: *Due to technical difficulties the double hyphen, which indicates the long options of GnuPG, in text embedded in paragraphs are represented individually as single hyphens. So please read double hyphens with long options.*

1.1 Public Key Encryption

Classic methods for encryption only use one key for encryption. The sender encrypts the message with this key. To be able to decrypt this the receiver needs to have this very same key. This key must have been given to the receiver in a way, that others won't have had the opportunity to obtain this key. If somebody else does have the key, this method of encryption is useless.

The use of so-called Public Keys can solve this problem. Public Keys is a concept where two keys are involved. One key is a Public Key that can be spread through all sorts of media and may be obtained by anyone. The other key is the Private Key. This key is secret and cannot be spread. This key is only available to the owner. When the system is well implemented the secret key cannot be derived from the public key. Now the sender will crypt the message with the public key belonging to the receiver. Then decryption will be done with the secret key of the receiver.

Crucial in this concept is that the secret key remains a secret and should not be given away or become available to anyone else but the owner of this key. YOU CANNOT SEND THIS KEY OVER THE INTERNET. Also it is very unwise to use GnuPG over `telnet` (you might consider never to use telnet based on the high security risks).

1.2 Digital Signatures

In order to prove that a message was really sent by the alleged sender the concept of Digital Signatures was invented. As the name says a message is digitally signed by the sender. By using this signature you can check the authenticity of a message. Using this will reduce the risk for Trojan horses (a message that claims to be a patch to a certain problem but actually contains a virus or does something bad with data on your computer). Also information or data can be verified as coming from a legitimate source and thus be regarded as real.

A digital signature is made through a combination of the secret key and the text. Using the senders public key the message can be verified. Not only will be checked if the correct sender is involved, also the content will be checked. So you know that the message comes from the sender and has not been changed during the transportation process.

1.3 Web of trust

A weak point of the Public key algorithms is the spreading of the public keys. A user could bring a public key with false user ID in circulation. If with this particular key messages are made, the intruder can decode and read the messages. If the intruder passes it on then still with a genuine public key coded to the actual recipient, this attack is not noticeable.

The PGP solution (and because of that automatically the GnuPG solution) exists in signing codes. A public key can be signed by other people. This signature acknowledges that the key used by the UID (User Identification) actually belongs to the person it claims to be. It is then up to the user of GnuPG how far the trust in the signature goes. You can consider a key as trustworthy when you trust the sender of the key and you know for sure that the key really belongs to that person. Only when you can trust the key of the signer, you can trust the signature. To be absolutely positive that the key is correct you have to compare the finger print over reliable channels before giving absolute trust.

1.4 Boundaries to security

If you have data that you would like to remain confidential, there is more to it than just determining which encoding algorithm to use. You should be thinking about your system security in general. Basically we consider PGP to be secure and as I write this documents no incidents of PGP being cracked are known to me. But that doesn't mean that all encoding must be safe then (for instance the NSA wouldn't notify me if they cracked PGP somehow, neither would other people who crack for real malicious grounds). But even if the PGP is fully 'unhackable', other means can be used to attack the security. Early February 1999 a Trojan Horse had been found that searched for secret PGP keys on the hard disk and FTP-ed them away. If the password has been chosen badly the secret key can easily be cracked.

Another technical possibility (although more difficult) is a Trojan Horse that broadcasts keyboard entries. Also possible (but very difficult) is to pass the content of a screen along. Then no cracking of scrambled messages needs to be done. For all these risks there need to be a good, well-thought security plan that is actually deployed.

It is not a goal to create paranoia among people, but to point out that a lot needs to be done to be more secure. The most important thing is to realize that encryption is just one step to security and is not a total solution. Trojan horses as they appeared in the Melissa virus in March 1999 proved that many companies are not prepared for that.

2 Installation

2.1 Sources for GnuPG.

The official download site is:

GnuPG Homepage `http://www.gnupg.org/download.html` . At that location you will find links to mirror sites.

Due to legal restrictions it is not allowed to download GnuPG from servers based in the USA. The USA imposes export restrictions on the export of cryptographic software. This is why PGP is always available in an international and a national (for the USA) version. For the international version the source code has been exported in a printed format as a book. In Europe (Oslo) it has been scanned. More information on that can be found on

International PGP Homepage `http://www.pgpi.com` . The international version of PGP is free to be imported into the United States as long as it is not reexported again.

If you already have an installed version of GnuPG or PGP, you should check the signature of the file (see 5 (Signatures)).

2.2 Configuration

You can obtain GnuPG as a Debian Package, as a RPM package (Redhat Package Manager) or in source code. GnuPG is included in the latest Fedora/Redhat Linux distributions. To check if you have GnuPG installed on your system, use:

```
rpm -q gnupg
```

The packages are installed as binary files with the tools needed for Linux platforms. When you need GnuPG for different platforms you need to compile this yourself. It would be appreciated when you make alternative installation methods for different platforms available to the general public.

Since development for the major part takes place with Linux (x86), translation to different systems should not be regarded as being a problem. The actual list of known operating systems that support GnuPG can be found on *GnuPG Homepage* . The procedure as described below is pretty platform independent. This procedure can be used to install GnuPG out of a source code tar-ball.

Unpack the tar-ball with the following commands, For sources compressed with gzip use:

```
tar xvzf gnupg-?.?.?.tar.gz
```

and for sources compressed with bzip2 use:

```
tar xvjf gnupg-?.?.?.tar.bz2
```

After the unpack, please step into the directory containing the source code. Then type

```
./configure
```

When doing this nothing special should happen. With

```
./configure --help
```

you can view the available configuration settings for compilation. If problems occur that have to do with internationalization (gettext), you can include a version that is delivered with the source code by using the option `-with-included-gettext` or switch it of by using the `-disable-NLS` option.

2.3 Compile

After this we want to compile the stuff by typing:

```
make
```

This should work without any problems. If any problems occur take the following steps (in the same order as described here): First try to solve this yourself (of course also by using the available documentation). Then make sure that your problem is not a known bug (check the BUGS file on http://www.gnupg.org). Then ask someone you know. If these steps do not solve your problem post a question to the GnuPG-mailing list (see 7 (Informationsources)). If the problem is path related, you should try `make clean`, then run `configure` again and retry to compile. If that doesn't work it is time to panic.

2.4 Installation

Now type:

```
make install
```

to actually copy the program and the man-pages into the installation directory. If you didn't change the installation directory when you did ./configure the `usr/local/share/gnupg/` will be the installation directory. You can find this directory in the file `options.skel`. When you change this `options.skel`. If you copy this to `~/.gnupg/options` the appropriate adjustments will be used as standard. Copying should occur automatically when creating `~/.gnupg/`. All possible options are well documented and explaining them here would not be useful.

You can run GnuPG as suid root. So the program runs with all the rights the superuser has. By doing this you exclude the possibility that certain parts of the program are stored externally and then could be read by anyone. It is not feasible for me to judge on the value of this risk. But running the program as suid root one should be alert to the danger of Trojan horses. Since a Trojan horse running as superuser can damage an entire system. If for this reason (or any other reason) you choose not run GnuPG as root you can switch off the warning by setting `no-secmem-warning` in `~/.gnupg/options`.

3 Using keys

3.1 Creating a key

With

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

a new key-pair is created (key pair: secret and public key). The first question is which algorithm can be used. You can read more about algorithms in

PGP DH vs. RSA FAQ `http://www.samsimpson.com/cryptography/pgp/pgpfaq.html` or in 7.4 (Applied Cryptography). You can easily (and maybe you should - since it is used so widely) use DSA/ ElGamal. This is not patented.

The next question is key length. This is something that is very user dependent. You need to choose between security and calculating time. If a key is longer the risk for cracking the message when intercepted decreases. But with a larger key calculation time also increases. If computing time is an issue you still should consider that you want to use the key for sometime. We all know that arithmetic performance increases very quickly, since new processors are getting quicker and quicker. So keep this in mind. The minimal key length GnuPG demands is 768 bits. However some people say you should have at a key-size of 2048 bits (which is also really a maximum with GnuPG at this moment). For DSA 1024 is a standard size. When security is a top priority and performance is less an issue you ought to pick the largest key-size available.

The system now asks to enter names, comment and e-mail address. Based upon the entries here the code is calculated. You can change these settings later. See 3.5 (Administering keypairs).

Finally you have to enter a password (actually passphrase would be more appropriate, since blanks are allowed). This password is used to be able to use the functionality which belongs to your secret key. A good passphrase contains the following elements:

- it is long,
- it has special (non alphanumeric) characters,
- it is something special (not a name),
- it is very hard to guess (so NOT names, birth dates, phone numbers, number of a credit card/checking account, names and number of children, ...)

By sometimes using CaPItaLs aNd SOMEtimes NoT you can build in further security. When you make your password make sure that you WILL NOT FORGET it. Since if you do messages are not legible and the use of your secret key has gone. It might be wise to generate some kind of a certificate containing this information (of course be careful that nobody gets to your passphrase). See 3.4 (Revoke).

After everything was entered the systems starts generating the keys. This will take some time. During that time it needs to collect a lot of random data. By working in a different screen you can help the system collecting changing random data. As you understand by now, the key will be always different. If you generate a key now and 5 minutes later with exactly the same data, you'll get two different keys. Now you must understand why you shouldn't forget your password.

3.2 Exporting keys

The command for exporting a key for a user is:

```
gpg --export [UID]
```

If no UID has been submitted all present keys will be exported. By default the output is set to `stdout`. But with the `-o` option this is sent to a file. It may be advisable using the option `-a` to write the key to a 7-bit ASCII file instead of a binary file.

By exporting public keys you can broaden your horizon. Others can start contacting you securely. This can be done by publishing it on your homepage, by finger, through a key server like http://www.pca.dfn.de/dfnpca/pgpkserv/ or any other method you can think of.

3.3 Importing keys

When you received someone's public key (or several public keys) you have to add them to your key database in order to be able to use them. To import into the database the command looks like this:

```
gpg --import [Filename]
```

if the filename is omitted the data will be read from `stdin`.

3.4 Revoke a key

For several reasons you may want to revoke an existing key. For instance: the secret key has been stolen or became available to the wrong people, the UID has been changed, the key is not large enough anymore, etc. In all these cases the command to revoke the key is:

```
gpg --gen-revoke
```

This creates a revocation certificate. *To be able to do this, you need a secret key*, else anyone could revoke your certificate. This has one disadvantage. If I do not know the passphrase the key has become useless. But I cannot revoke the key! To overcome this problem it is wise to create a revoke license when you create a key pair. And if you do so, keep it safe! This can be on disk, paper, etc. Make sure that this certificate will not fall into wrong hands!!!! If you don't someone else can issue the revoke certificate for your key and make it useless.

3.5 Key administration

With the GnuPG system comes a file that acts as some kind of database. In this file all data regarding keys with the information that comes with the keys is stored (everything until the Ownertrust values: for more information on that read 3.6 (Key signing)). With

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

all present keys will be displayed. To see the signatures as well type:

```
gpg --list-sigs
```

(see 3.6 (Key signing) for further information). To see the fingerprints type:

```
gpg --fingerprint
```

You want to see "Fingerprints" to ensure that somebody is really the person they claim (like in a telephone call). This command will result in a list of relatively small numbers.

To list the secret keys you type:

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

Note that listing fingerprints and signatures from private keys has no use what soever.

In order to delete a public key you type:

```
gpg --delete-key UID
```

For deleting a secrete key you type:

```
gpg --delete-secret-key
```

There is one more important command that is relevant for working with keys.

```
gpg --edit-key UID
```

Using this you can edit (among other things) the expiration date, add a fingerprint and sing your key. Although it is too logic to mention. For this you need your passphrase. When entering this you will see a command line.

3.6 Key signing

As mentioned before in the introduction there is one major Achilles' heel in the system. This is the authenticity of public keys. If you have a wrong public key you can say bye bye to the value of your encryption. To overcome such risks there is a possibility of signing keys. In that case you place your signature over the key, so that you are absolutely positive that this key is valid. This leads to the situation where the signature acknowledges that the user ID mentioned in the key is actually the owner of that key. With that reassurance you can start encrypting.

Using the `gpg -edit-key UID` command for the key that needs to be signed you can sign it with the `sign` command.

You should only sign a key as being authentic when you are ABSOLUTELY SURE that the key is really authentic!!!. So if you are positive you got the key yourself (like on a key signing party) or you got the key through other means and checked it (for instance by phone) using the fingerprint-mechanism. You should never sign a key based on any assumption.

Based on the available signatures and "ownertrusts" GnuPG determines the validity of keys. Ownertrust is a value that the owner of a key uses to determine the level of trust for a certain key. The values are

- 1 = Don't know
- 2 = I do NOT trust
- 3 = I trust marginally
- 4 = I trust fully

If the user does not trust a signature it can say so and thus disregard the signature. Trust information is not stored in the same file as the keys, but in a separate file.

4 Encrypt and decrypt

After installing everything and configuring everything in the way we want, we can start on encrypting and decrypting.

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 UID` or by using the option `-local-user UID`. This causes the default key to use to be replaced by wanted key.

If you want to change recipient this can be done by the option `-r` or by the option `-recipient`.

4.1 Encrypt

The command to encrypt is

```
gpg -e Recipient [Data]
```

or

```
gpg --encrypt Recipient [Data]
```

To avoid the risk that somebody else claims to be you, it is very useful to sign everything you encrypt, see 5 (signatures).

4.2 Decrypt

The command for decrypting is:

```
gpg [-d] [Data]
```

or

```
gpg [--decrypt] [Data]
```

Also here `stdout` is preset, but with the `-o` option you can redirect the output to a file.

5 Signing and checking signatures

To sign data with your own key, use the command:

```
gpg -s (or --sign) [Data]
```

By doing this also compression takes place. This means that the result is not legible. If you want a legible result you can use:

```
gpg --clearsign [Data]
```

this will make sure that the results are clearly legible. Furthermore it does the same (signing data).

With

```
gpg -b (or --detach-sign) [Data]
```

you can write the signature in a separate file. It is highly recommended to use this option especially when signing binary files (like archives for instance). Also the `-armor` option can be extremely useful here.

Quite often you find that data is encrypted and signed as well. The full instruction looks like:

```
gpg [-u Sender] [-r Recipient] [--armor] --sign --encrypt [Data]
```

The functionality of the options `-u` (`-local-user`) and `-r` (`-recipient`) are as described before.

When encrypted data has been signed as well, the signature is checked when the data is decrypted. You can check the signature of signed data by using the command:

```
gpg [--verify] [Data]
```

This will only work (of course) when you own the public key of the sender.

6 Front ends

To make life a lot easier in using GnuPG, you have a wide choice of programs that either use or support GnuPG encryption. There are graphical front ends that put your key administration at the click of a mouse button and many MUAs (Mail User Agents) let you encrypt and sign your Email messages seamlessly. A nearly full list of front ends is available from the *GnuPG Frontends* page. We will highlight a few of them in this section.

6.1 Graphical interfaces

6.1.1 GPA

GPA , the *GNU Privacy Assistant* is a graphical user interface for the GNU Privacy Guard (GnuPG). This is the standard graphical front end, provided by the GnuPG project. With GPA, you can view your keyring, import and export keys, generate keys, edit key properties and encrypt, sign or decrypt documents. Installing GPA is easy. Download the tarball, unpack and do the usual

```
./configure; make; make install.
```

Start the program by typing

```
gpa
```

6.1.2 Seahorse

Seahorse

is a GNOME front-end for GnuPG. It can be used for sign, encrypt, verify and decrypt text and files. The text can be taken from the clipboard, or written directly in the little editor it has. Seahorse is also a key manager, which can be used to edit almost all the properties of the keys stored in your key rings. You can install Seahorse from a Debian package (RPMs are not available at this time) or from the source tarball. Installing from source is like any other package. Download, untar, configure and make install. The installation puts seahorse in `/usr/local` and puts a menu item in the Gnome 'Applications' menu.

6.1.3 Geheimnis

KGPG is a front end for GnuPG which is based upon KDE. KGPG supports key signing, importing and exporting. It can also work with other KDE tools interactively like konquerer.

6.2 Email programs

Most popular Email programs (or MUAs) support GnuPG. Among these are at least the following:

- Mozilla
- Evolution
- Pine
- Kmail
- Eudora
- Mutt
- exmh

There are probably more; it is hardly possible to try them all.

Using GnuPG support in your mail program lets you decrypt Email messages sent to you that are encrypted with your public key, sign your messages so the receiving party can make sure you are the author and encrypt your Email with the public keys of your recipients.

6.2.1 Mozilla and Enigmail

Mozilla does not have GnuPG support in itself. To use GnuPG encryption with Mozilla, you must install a plug-in, such as *EnigMail* . Enigmail is a "plugin" for Mozilla/Netscape Mail which allows users to access the authentication and encryption features provided by the popular GPG and PGP software. Enigmail can encrypt/sign mail when sending, and also decrypt/authenticate received mail. It can also import/export public keys.

Enigmail can easily be installed by Mozilla extension management. The only thing you should do is to click on the proper link related with your Mozilla version on the *Download* page. Unfortunately, RPMs for enigmail are not available.

6.2.2 Evolution

Evolution is a well-known MUA for Linux and has fairly good GnuPG support. It can get e-mails encrypted and decrypt them. It also has built in key signing and key authorising options. It supports S/MIME but inline pgp support (which is not standard but used by some MUAs) is not implemented. To configure the GnuPG settings you should check the mail account preferences.

6.2.3 Kmail

Kmail, the standard Email program for KDE has integrated support for GnuPG and PGP encryption. To set things up so you can sign and decrypt messages, you have to enter your GnuPG user ID in the 'Identity' section of the Kmail configuration. When you send a new message, the message will not be signed or encrypted by default. You have to check the 'Sign message' and 'Encrypt message' buttons in the tool bar.

7 Sources of information

7.1 GnuPG

- The *GnuPG Homepage* `http://www.gnupg.org`
- The GnuPG Mailing list, including archives and descriptions on the *GnuPG Homepage* .
- The information enclosed in the GnuPG project (updated until version 0.9.2), however not yet very extensively done. And not to forget:

  ```
  gpg --help
  ```

  . This is very valuable information.

7.2 PGP

PGP is the older and (still) widely spread and used cryptography program. Through the years a lot of documents have been made. This can be considered as very useful information. A lot of that information is so general that you can apply that to GnuPG as well. Check out the following URLs for these documents:

- The *International PGP Homepage* `http://www.pgpi.com`

- The *PGP DH vs. RSA FAQ* `http://www.scramdisk.clara.net/pgpfaq.html` has information on the differences of these two algorithms. These are the two algorithms used by GnuPG.

7.3 Keyservers

- *Keyserver.net* `http://www.keyserver.net`
- `http://wwwkeys.eu.pgp.net`

7.4 Books

- B. Schneier, "Applied Cryptography, Second Edition", Wiley, 1996 Deutsche Ausgabe unter dem Titel "Angewandte Kryptographie", Addison-Wesley, 1996

8 About this document

8.1 Versions

Original German versions: Version 0.1 was the first version in German

Version 0.1.0 (English) April 30th 1999, Dutch Queen's Day.

- This version is the translation of the German version in English with some adjustments.

Version 0.1.1 (German)

- New section "Boundaries to security"
- Improved explanation of signatures
- Changes after comments from Werner Koch (thanks!)

Version 0.1.2 (English) April 3, 2002

- Corrected a few typos.
- New section about front ends.

Version 0.1.3 (Dutch) May 17, 2002

- This version is a translation of the English version into Dutch.

Version 0.1.4 (Turkish) May 3, 2004

- This version is a translation of the English version into Turkish. Also some additions are made on English version. Broken links are fixed. Frontends are revised.

All changes are documented in a diff file:

dieses Dokument `http://www.stud.uni-goettingen.de/~s070674/GnuPGMiniHowto/`

For the English or Dutch version: All remarks for this document can be sent to Brenno J.S.A.A.F. de Winter (*brenno@dewinter.com*). or Arjen Baart (*arjen@andromeda.nl*). Comments help us make a better document and are greatly appreciated.

For the German version: Anregungen, Kritik, Verbesserungen und Erweiterungen einfach an Michael Fischer v. Mollard (*fischer@math.uni-goettingen.de*) senden, damit dieses Dokument weiter verbessert werden kann.

For the Turkish version: Baris Cicek (*baris teamforce.name.tr*)