Using GnuPG Keys

Key Signing Party

Modern mail clients can handle GnuPG encryption and decryption with very little intervention. Unfortunately, you do need to set up the keys – and that is a task that does not fall far short of rocket science.

BY PATRICIA JUNG

Fritz von Beust, visipix.com

All those resolutions! By now, not only the mail gurus should be aware that email messages are readable for anyone with access to the data in transit to their final destination (and on the target server). Although one tends to use an envelope to protect the confidentiality of even the most harmless of written messages, people tend to hide their guilty conscience with respect to their half-hearted approach to **PGP or GnuPG keys** behind statements such as "I have nothing to hide from my provider or the Feds"

Nearly any email program will support electronic envelopes of this kind. Although there is a modicum of truth in the statement, "never used it, because it is too complex", a total embargo on PGP/GnuPG is definitely not justifiable. And if friends or colleagues then invite you to a **Key Signing** party, you can expect your guilty conscience to win out in the end…

Tools

Assuming that you have installed the GnuPG command line tool *gpg*, the *gpg --gen-key* ("*gen*erate *key*") command will create a new key-pair for your own use. If you are not sure of the syntax, typing

```
gpg --help | less
```

will provide a quick overview; at least there is no lack of documentation for this aspect. If you consider yourself command line challenged, you might like to look into "*G*NU *P*rivacy *A*ssistant" *gpa* (*http://www.gnupg.org/related_software/gpa/*), which is a GUI version of the same tools (see Figure 1).

Unfortunately, this tool provides only a fraction of the command line tool's functionality at present; also, only a few distributions supply the latest version, and the GUI of the older versions is a challenge in itself. Reason enough not to discuss it any further at this point.

A few encryption and signing tasks, just to get the hang of the new tool, including configuring your mail program, signing a few messages, encrypting them with your own private key and sending them to yourself – then it's time to get ready for the party.

The first important point: Before your friends and colleagues can sign your public key (and, more importantly, use it to create envelopes for messages they send to you), they first need your key. The traditional method of extracting it is as follows

```
gpg --export -a ➥
"Name" >public_key
```

This writes the *A*SCII text key to a file called *public_key* that you can use to distribute your key. But who wants to go to the trouble of sending the key to all the party-goers (whose email addresses you may not even know), or even distributing the key on floppies? Anyone who missed the party and wants to send you encrypted messages, would still need your public key.

Fortunately, there is such a thing as a PGP/GnuPG key-server, and *gpg* facilitates uploading your public key to the server (and downloading the keys of potential mail correspondents). Many of these servers synchronize their key databases, and this typically allows you to select just one key-server, such as the one run by MIT. You would then enter the server in your GnuPG configuration file *~/.gnupg/options* (this could be called *gnupg.conf* instead)

THE AUTHOR

Patricia Jung has spent her working life as an editor, technical writer and system administrator on Linux and FreeBSD systems. Having been an editor for Linux Magazine for quite a while now she feels privileged to continue building a career based on free Unix systems exclusively.

```
keyserver pgp.mit.edu
```

… and use the *gpg --search-keys email@address* command to add the key for the mail address you supplied to your public keyring (see Listing 1). This file is stored along with your private keyring, *secring.gpg*, in the *~/.gnupg* directory as *pubring.gpg*.

If the search key you entered matches multiple keys on the server, *gpg* displays a list like the one shown in our example. Instead of responding with "import keys *1-5*", you can simply type *1* (or *2*, or …) at this point: *gpg* displays the key ID for each selection – this is *1E786A45* in every case in Listing 1. In other words, the same key is being used for multiple email addresses.

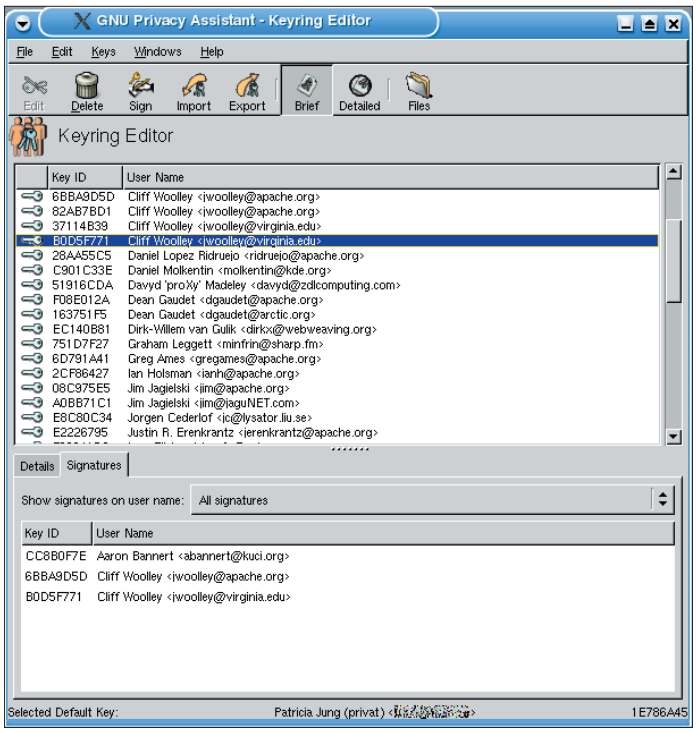

Figure 1: The GNU Privacy Assistant only provides a fraction of GnuPG's functionality at present

However, before you attempt to upload your own public key to the key-server, using *gpg --send-keys "Name"*, first take a deep breath! You will need to protect your own private key zealously from this point on.

If you forget the passphrase, there is no way to revoke your key. This is why it makes sense to use *gpg --gen-revoc "name"* to create a revocation certificate for your own key, and file a hard copy in a safe place, in case disaster strikes. You should do this now, while the passphrase is still fresh in your mind and before you upload the public key.

Fingerprints

You, and the people you exchange messages with, will need a way of verifying that the key stored on the server really does contain the data it should. To do that, you should create a fingerprint of your own public key, by typing *gpg --fingerprint "name"* or *gpg --fingerprint email@address* (most, but not all *gpg* command line options allow you to identify the required key either by the matching name, the key ID, the email address, or even its **local part** – provided the information supplied is unique).

Just to be sure, you might like to store a copy of your fingerprint. Also, you can configure your mail client to dispatch the fingerprint in a user-defined header (a so-called X header) with every outgoing message (such as in: *X-GPG-Fingerprint:*). Remember that everything depends on the fingerprint: to ensure that a key that you downloaded from the key-server really is what the key owner intended, you can ask *gpg* for the fingerprint of the keyring entry, and check this series of **hexadecimal numbers** in the output from…

```
webmistress@fri:~$ gpg ➥
--fingerprint ➥
pjung@linux-user.de
[...]
Key fingerprint = 2350 ➥
B799 81E8 B20B 3743 ➥
D541 CA1E C447 1E78 6A45
```

… against the fingerprint distributed by the key owner.

This is why you will need to print enough hard copies of your own fingerprint to distribute to the party guests. At some parties, fingerprints are collected before the event and passed on to each of the attendees.

It's Party Time

Unfortunately, comparing fingerprints only lets you verify the key itself, but you have no way of knowing if the key owner really is the person whose name is stored for the key unless you have that person present some ID. So anyone attending the signing party will need

GLOSSARY

PGP/GnuPG Keys: *The "GNU Privacy Guard" (http://www.gnupg.org/) is a patent-free Open Source implementation of the OpenPGP standards, which in turn is based on the "Pretty Good Privacy" tool (now turned commercial). If you want to use GnuPG or PGP to encrypt or sign documents, you will need a key-pair comprising a private key (which needs to be kept secret and protected by a passphrase), and a public key (which is distributed to others).*
When you send a message to someone with a GnuPG or PGP envelope, you first use the recipient's public key to encrypt the message. The recipient's private key is required to decrypt the message. When you sign a document, you utilise your own private key. The recipient can then decrypt the message use the sender's public to verify that the message has reached its final destination without being manipulated or damaged en route.
Key Signing: *Key signing refers to signing a correspondence partner's public key with your own private key, thus validating the link between the key owner and her key, and building up a "Web of Trust", a trusted network: If the public key of a third party has been signed by someone I trust, I can assume that the person I trust has taken care to verify the credentials in question i.e. the link between the third party's identity and key. This in turn allows me to regard the third party's key as trustworthy.*
Local part: *The part of an email address left of the @ sign. Traditionally, this was always the username on a mail system, but today it only means that this part of the name is local to the computer or domain (the part of the name to the right of the @ sign).*
Hexadecimal numbers: *A base 16 numeric system (instead of base 10 in the case of the normal decimal system) that uses the numbers 0-9, followed by A (which corresponds to 10) through F (15).*

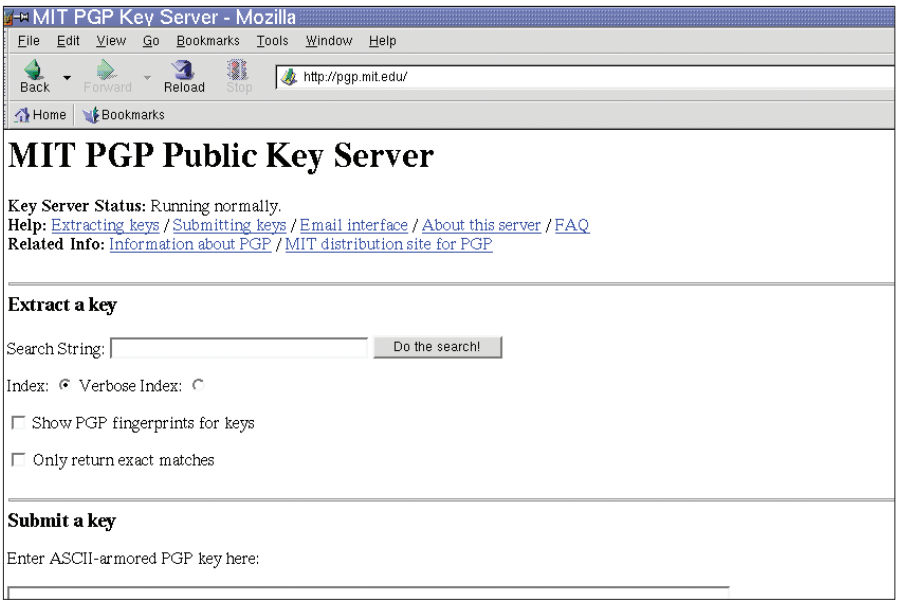

Figure 2: You can use a Web interface to query the MIT key-server

some kind of ID – a passport, ID card, or some other written documents – how else could a stranger trust, Patricia Jung to really be Patricia Jung? After convincing yourself that the person who handed out the hard copy of the fingerprint really does match the person depicted in the ID they present, you can go on to download the public key and compare its fingerprint with the one you were given. Now you have validated both the key and the correspondent's identity, but how can you be sure that mail addressed to the email address indicated by the key will really reach the person you validated?

To allow them to be added or revoked during the lifetime of a key, addresses do not have any influence on the fingerprint (this also applies to key signatures). So if you really want to ensure that the correspondent, her email address and key really do match before you add your signature, you will first need to check the mail addresses.

To do so, you send a message encrypted using the public key you validated to the correspondent at the address that you have been asked to sign. The message must contain a request to sign the content of the message and return it. To ensure that the recipient cannot just guess the content of the challenge, but really has to decrypt it, the body of the mail includes a ran-

Listing 1: Importing a public key from a key-server

```
webmistress@fri:~$ gpg search-keys ➥
pjung@linux-user.de
gpg: WARNING: Confidential data may be stored ➥
on disk.
gpg: see http://www.gnupg.org/documentation➥
/faqs.html for more information
gpg: created keyring ➥
'/home/webmistress/.gnupg/pubring.gpg'
gpg: searching for "pjung@linux-user.de" on HKP ➥
server pgp.mit.edu
Keys 1-5 of 5 for "pjung@linux-user.de"
(1)     Patricia Jung <pjung@linux-user.de>
          1024 bit DSA key 1E786A45, created ➥
2003-01-17
(2)     Patricia Jung <pjung@linux-magazin.de>
          1024 bit DSA key 1E786A45, created ➥
2003-01-17
[...]
(5)     Patricia Jung <pjung@linux-magazine.com>
          1024 bit DSA key 1E786A45, created ➥
2003-01-17
Enter number(s), N)ext, or Q)uit > 1-5
gpg: Key 1E786A45: "Public key "Patricia Jung
<pjung@linux-user.de>" imported
gpg: Number of keys processed: 1
gpg:                              imported: 1
```

dom string, which can be generated using the following command:

```
dd if=/dev/random bs=1 count=10➥
 2>/dev/null | uuencode -m - | ➥
sed -n 2p
```

Some scripts that help automate this procedure – assuming some customization – are available on the Web from [1], for example.

When the random string is returned to you (and assuming you made a note of it), you know that the recipient can decrypt the message, and if the signature attached to the reply is okay, you also know that the message has not been manipulated en route.

And now the signing ceremony can begin. Run *gpg --edit email@address* to do so; this will open an interactive tool that displays a list of addresses assigned to this key (see Listing 2). You can type *help* [Enter] at the internal prompt, to display your current options. If you want to sign all of these addresses, you can simply type *sign* [Enter]; if you want to apply restrictions, select the appropriate number before entering a command, and press then [Enter] to confirm. In the first case *gpg* will prompt you to confirm (Do you really want to sign all of these addresses?) – you can then type [y] for yes, or [n] for no. If there is only one email address to sign, then this question is skipped.

Then the question of trust: Have you really "done very careful checking?" (answer [3]) or have you simply "done casual checking ([2])?". This specifies the level of trust that you will be assigning to the signature for this key. Again you are prompted to confirm, and then the moment of truth: the tool asks you to supply the passphrase for your own secret key. Typing the passphrase returns you to the internal prompt where you can type *quit*. Make sure that you confirm when prompted to save your changes; *gpg* will not save the third party key with your signature attached, unless you do.

Of course, none of this is of much use, unless the signature is published, because other mail users who know the signer – that is, you – but do not know the key owner, will base the trust they place in the key on the trust relationship between the signer and the keyholder. The signer has the option of re-exporting the signed key and sending it back to the owner (and placing the burden of distributing the key on the owner), or using *gpg --send-keys "name"* to upload the signature to the key-server. You can then type *gpg --refresh-keys* to update your keyring – and make your bid for a place in the Top 1000 of the Web of Trust [2]. ■

Listing 2: Signing a third party key

```
webmistress@fri:~$ gpg --edit pjung@linux-user.de
gpg (GnuPG) 1.2.1; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/1E786A45  created: 2003-01-17 expires: never      trust: u/u
sub  2048g/62D4F0F4  created: 2003-01-17 expires: never
(1)  Patricia Jung <pjung@linux-user.de>
(2)  Patricia Jung <pjung@linux-magazin.de>
[...]
(5). Patricia Jung <pjung@linux-magazine.com>

Command> 1
pub  1024D/1E786A45  created: 2003-01-17 expires: never      trust: u/u
sub  2048g/62D4F0F4  created: 2003-01-17 expires: never
(1)* Patricia Jung <pjung@linux-user.de>
(2)  Patricia Jung <pjung@linux-magazin.de>
[...]
(5). Patricia Jung <pjung@linux-magazine.com>

Command> sign
pub  1024D/1E786A45  created: 2003-01-17 expires: never      trust: u/u
 Primary key fingerprint: 2350 B799 81E8 B20B 3743  D541 CA1E C447 ➥
1E78 6A45

     Patricia Jung <pjung@linux-user.de>

How carefully have you verified the key you are about to sign actually➥
 belongs
to the person named above?  If you don't know what to answer, enter "0".
   (0) I will not answer. (default)
   (1) I have not checked at all.
   (2) I have done casual checking.
   (3) I have done very careful checking.

Your selection? 3
Are you really sure that you want to sign this key
with your key: "Webmistress <webmistress@answergirl.de>"

I have checked this key very carefully.

Really sign? y
You need a passphrase to unlock the secret key for
user: "Webmistress <webmistress@answergirl.de>"
1024-bit DSA key, ID 2F0F137E, created 2003-04-28
secret passphrase of your own secret key
Command> quit
Save changes? y
```

INFO

[1] Automating the Challenge Response Process: *http://www.lysator.liu.se/~jc/verifygpgmail.sh*

[2] Top 1000: *http://skylane.kjsl.com/~jharris/ka/2003-07-27/top1000table.html*